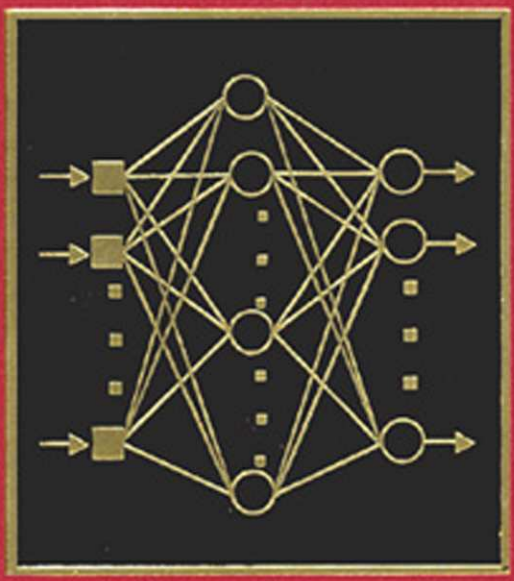
ELSEVIER
ACADEMIC
PRESS

Advances in COMPUTERS

Volume 61

Architectural Issues



Edited by

MARVIN V. ZELKOWITZ

Advances

in COMPUTERS
VOLUME 61

This Page Intentionally Left Blank

Advances in
COMPUTERS

Architectural Issues

EDITED BY

MARVIN V. ZELKOWITZ

Department of Computer Science
and Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland

VOLUME 61

ELSEVIER
ACADEMIC
PRESS

Amsterdam Boston Heidelberg London New York Oxford
Paris San Diego San Francisco Singapore Sydney Tokyo

ELSEVIER B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam
The Netherlands

ELSEVIER Inc.
525 B Street, Suite 1900
San Diego, CA 92101-4495
USA

ELSEVIER Ltd
The Boulevard, Langford Lane
Kidlington, Oxford OX5 1GB, UK

ELSEVIER Ltd
84 Theobalds Road
London WC1X 8RR, UK

© 2004 Elsevier Inc. All rights reserved.
This work is protected under copyright by Elsevier Inc., and the following terms and conditions apply to its use:

Photocopying
Single photocopies of single chapters may be made for personal use as allowed by national copyright laws. Permission of the Publisher and payment of a fee is required for all other photocopying, including multiple or systematic copying, copying for advertising or promotional purposes, resale, and all forms of document delivery. Special rates are available for educational institutions that wish to make photocopies for non-profit educational classroom use.

Permissions may be sought directly from Elsevier's Rights Department in Oxford, UK: phone (+44) 1865 843830, fax (+44) 1865 853333, e-mail: permissions@elsevier.com. Requests may also be completed on-line via the Elsevier homepage (http://www.elsevier.com/locate/permissions).

In the USA, users may clear permissions and make payments through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA; phone: (+1) (978) 7508400, fax: (+1) (978) 7504744, and in the UK through the Copyright Licensing Agency Rapid Clearance Service (CLARCS), 90 Tottenham Court Road, London W1P 0LP, UK; phone: (+44) 20 7631 5555, fax: (+44) 20 7631 5500. Other countries may have a local reprographic rights agency for payments.

Derivative Works
Tables of contents may be reproduced for internal circulation, but permission of the Publisher is required for external resale or distribution of such material. Permission of the Publisher is required for all other derivative works, including compilations and translations.

Electronic Storage or Usage
Permission of the Publisher is required to store or use electronically any material contained in this work, including any chapter or part of a chapter.

Except as outlined above, no part of this work may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without prior written permission of the Publisher.

Address permissions requests to: Elsevier's Rights Department, at the fax and e-mail addresses noted above.

Notice
No responsibility is assumed by the Publisher for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions or ideas contained in the material herein. Because of rapid advances in the medical sciences, in particular, independent verification of diagnoses and drug dosages should be made.

First edition 2004

Library of Congress Cataloging in Publication Data
A catalog record is available from the Library of Congress.

British Library Cataloguing in Publication Data
A catalogue record is available from the British Library.

ISBN: 0-12-012161-1
ISSN (Series): 0065-2458

♾ The paper used in this publication meets the requirements of ANSI/NISO Z39.48-1992 (Permanence of Paper).

Printed in Great Britain.

Contents

Security Issues and Solutions in Distributed Heterogeneous Mobile Database Systems

A.R. Hurson, J. Ploskonka, Y. Jiao, and H. Haridas

Disruptive Technologies and Their Affect on Global Telecommunications

Stan McClellan, Stephen Low, and Wai-Tian Tan

Ions, Atoms, and Bits: An Architectural Approach to Quantum Computing

Dean Copsey, Mark Oskin, and Frederic T. Chong

This Page Intentionally Left Blank

Contributors

Koen De Bosschere is a Professor at the Faculty of Applied Sciences of Ghent University, where he teaches courses in computer architecture, operating systems and declarative programming languages. His research interests include logic programming, system programming, parallelism and debugging. De Bosschere received his PhD in Applied Sciences from Ghent University in 1992. Contact him at kdb@elis.UGent.be.

Frederic T. Chong is an Associate Professor of Computer Science and a Chancellor's Fellow at the University of California at Davis. He received his BS in 1990, MS in 1992, and PhD in 1996 in Electrical Engineering and Computer Science from MIT and is a recipient of the National Science Foundation's CAREER award. His current work focuses on architectures and applications for novel computing technologies.

Dean Copsey is a doctoral student in Computer Science at the University of California at Davis. He received his BS in 1986 in Chemical Engineering and his MS in 1996 in Computer Science, both from the University of California at Davis. His current research interests include quantum-computing architectures, fault tolerance in nanoscale transistor technologies, and tile architectures for digital signal processing.

Patricia Costa is a scientist at the Fraunhofer Center for Experimental Software Engineering, Maryland. She has a B.Sc. in 1996 and a M.Sc. in 1999 in Computer Science from the Federal University of Minas Gerais, Brazil and a M.Sc. in Telecommunications Management in 2001 from University of Maryland University College. Her research interests include software architecture, agile methods and knowledge management. Contact her at pcosta@fc-md.umd.edu.

Lieven Eeckhout is a Postdoctoral Fellow of the Fund for Scientific Research—Flanders (Belgium) (F.W.O. Vlaanderen) in the Department of Electronics and Information Systems at Ghent University, Belgium. His research interests include computer architecture, performance analysis and workload characterization. He obtained his PhD in Computer Science and Engineering from Ghent University in December 2002. Contact him at leeckhou@elis.UGent.be.

H. Haridas received his Masters degree in Computer Science and Engineering from the Pennsylvania State University in May 2003 and joined the Automation and Con-

trol IS department at Honeywell as an engineer. His thesis at Penn State covers issues in Wireless medium, Security, and Multi-databases. Harshal received his bachelor's degree in Computer Engineering from Pune Institute of Computer Technology.

A.R. Hurson is a Computer Science and Engineering professor at The Pennsylvania State University. NSF, NCR CORP., DARPA, IBM, Lockheed Martin, ONR, and Penn State University have supported his research for the past 20 years. He has published over 200 technical papers in areas including database systems, multidatabases, object oriented databases, security, mobile computing environment, computer architecture and cache memory, parallel and distributed processing, dataflow architectures, and VLSI algorithms.

Y. Jiao is a Ph.D. candidate at the Computer Science Department, Pennsylvania State University. She received the B.Sc. degree from Civil Aviation Institute of China, Tianjin, China and the M.Sc. degree from the Pennsylvania State University, State College, Pennsylvania, in 1997 and 2002, respectively, both in computer science. Her main research interests are mobile data access system security, mobile agent security, and energy-efficient information system design.

Mikael Lindvall is a scientist at Fraunhofer Center for Experimental Software Engineering, Maryland. He specializes in work on software architecture and impact analysis as well as experience and knowledge management in software engineering. He is currently working on defining cost-efficient approaches to evaluate software architectures. He received his PhD in Computer Science from Linköpings University, Sweden. His PhD work was based on a commercial development project at Ericsson Radio and focused on the evolution of object-oriented systems. Contact him at mlindvall@fc-md.umd.edu.

Stephen Low is a Distinguished Technologist in the Carrier Grade Systems group at Hewlett Packard. During his 12 years at HP, Compaq and Tandem, he has been a principal architect of several high availability platforms used in core telecommunications applications. He holds several patents in this area and has multiple publications. Prior to joining HP, he was a member of technical staff at AT&T Bell Labs, GPT and Northern Telecom. Dr. Low holds a Ph.D. in theoretical physics from the University of Texas at Austin.

Stan McClellan is a Distinguished Technologist in the Carrier-Grade Systems group at Hewlett Packard. In this capacity, he is responsible for the architectural definition of next-generation communication systems. Prior to joining HP, he held various positions in aerospace, academia, and telecommunications industries. Dr. McClellan holds a Ph.D. in Electrical Engineering from Texas A&M University.

Mark Oskin is an Assistant Professor at the University of Washington in Computer Science and Engineering. His current research interests include quantum architec-

tures and languages, and scalable execution substrates for future silicon technologies.

J. Ploskonka is a master's student at the Pennsylvania State University studying computer science and engineering. She received a bachelor's degree in electrical engineering from Grove City College. Her research interests include security and sensor networks.

Wai-tian Tan is a member of the Media Communications and Networking Department in HP Labs. He received the BS degree from Brown University in 1992, the MSEE degree from Stanford University in 1993 and the Ph.D. degree from U.C. Berkeley in 2000. He worked for Oracle Corporation from 1993 to 1995. His research focuses on adaptive media streaming, both at the end-point and inside the delivery infrastructure.

Roseanne Tesoriero Tvedt is a scientist at the Fraunhofer Center for Experimental Software Engineering, Maryland and President and founder of the software development company Pan Orange, Inc. She received Masters and Ph.D. degrees in Computer Science from the University of Maryland. Her research interests include software architecture evaluation, agile methods, knowledge management, and computer science education. She can be reached at rtvedt@fc-md.umd.edu or rtvedt@panorange.com.

This Page Intentionally Left Blank

Preface

In this volume of the **Advances in Computers** we present five chapters on computer architecture. The chapters discuss both how to develop new computing platforms that take advantage of new technology and how to evaluate the software that is being designed on those platforms. This series of books, the present one being volume 61, dates from 1960 and is the longest continuously published collection that chronicles the advances and ever changing computer landscape.

In Chapter 1, "Evaluating software architectures," Roseanne Tesoriero Tvedt, Patricia Costa, and Mikael Lindvall present a method for evaluating the designs of a software system. They present a seven-step process for evaluating a design looking for discrepancies between the abstracted design they develop and the system that has been implemented. Their process then suggests changes to the source code that will put the system in compliance with the design they develop. Having both synchronized should greatly aid in maintaining that system in the future. They present two case studies of using their process on existing products.

Lieven Eeckhout and Koen De Bosschere present "Efficient architectural design of high performance microprocessors" in Chapter 2. They present a new simulation methodology for designing high performance microprocessors by combining several recently proposed techniques, such as statistical simulation, representative workload design, trace sampling and reduced input sets. They show how simulation of these new designs can be made efficient by selecting a limited but representative workload, by applying trace sampling and reduced input sets to limit the simulation time per benchmark, and by optimizing the architectural simulators.

Chapter 3 contains "Security issues and solutions in distributed heterogeneous mobile data systems" by A. R. Hurson, J. Ploskonka, Y. Jiao and H. Haridas. With increased use of mobile computing resources and centralized databases, security of these systems is of growing importance. When wireless communication links and mobility are introduced, adapting existing methods may not be satisfactory, especially where mobile devices are already resource-poor. New techniques are needed to handle the constraints introduced by mobile systems.

In Chapter 4 "Disruptive technologies and their affect on global telecommunications" by Stan McClellan, Stephen Low, and Wai-Tian Tan, the authors categorize

several key technologies, platforms, and services which are affecting the evolution of telecommunications networks. With much standardization of these hardware, operating systems, and telecommunications services across the industry, they survey issues related to quality of service in packet-switched networks and some key protocols affecting the structure and operation of telephony networks.

As microprocessors have become faster and smaller, the ultimate size and speed reduction has been the dream of using atoms and molecules to store information. This is the realm of quantum computing. In the last chapter of this volume "Ions, atoms, and bits: An architectural approach to quantum computing," the authors Dean Copsey, Mark Oskin, and Frederick T. Chong discuss architectural issues in quantum computing. Unlike classical signals, quantum data cannot be transmitted over a wire, but must be move step-wise from location to adjacent location. They compare the swapping channel with a longer-range teleportation channel and discuss error correction, which is necessary to avoid data corruption.

I hope that you find this volume of use in your work or studies. If you have any suggestions of topics for future chapters, or if you wish to contribute such a chapter, I can be reached at mvz@cs.umd.edu

Marvin Zelkowitz
University of Maryland,
College Park, MD, USA

Evaluating Software Architectures

ROSEANNE TESORIERO TVEDT, PATRICIA COSTA, AND MIKAEL LINDVALL

Fraunhofer Center for Experimental Software Engineering
College Park, MD 20740
USA
rtvedt@fc-md.umd.edu
pcosta@fc-md.umd.edu
mlindvall@fc-md.umd.edu

Abstract

As software systems become increasingly complex, the need to investigate and evaluate them at high levels of abstraction becomes more important. When systems are very complex, evaluating the system from an architectural level is necessary in order to understand the structure and interrelationships among the components of the system. There are several existing approaches available for software architecture evaluation. Some of these techniques, pre-implementation software architectural evaluations, are performed before the system is implemented. Others, implementation-oriented software architectural evaluations, are performed after a version of the system has been implemented. This chapter briefly describes the concepts of software architecture and software architectural evaluations, describes a new process for software architectural evaluation, provides results from two case studies where this process was applied, and presents areas for future work.

ADVANCES IN COMPUTERS, VOL. 61
ISSN: 0065-2458/DOI 10.1016/S0065-2458(03)61001-6

Copyright © 2004 Elsevier Inc.
All rights reserved.

1. Introduction

In the early days of software development, much attention was given to issues related to the design of algorithms and data structures. This emphasis on algorithms and data structures was due in part to the fact that programming language features were closely related to the hardware and it was difficult to abstract and reason about systems at a higher level. As more sophisticated features were added to programming languages, it became easier to create more complex systems. As systems became increasingly complex, the ability to reason about systems at higher levels of abstraction became progressively more important. The shift in design concerns is evident in an early paper by Deremer and Kron [16] where the authors describe the difference between programming-in-the-small and programming-in-the-large and argued for a Module Interconnection Language (MIL) to help describe a system at an architectural level. Since that time, programming languages have become even more sophisticated and the use of networked components has become more prevalent. With ever increasing system complexity, the need to examine and evaluate systems at higher levels of abstraction becomes essential.

Software architectural evaluation techniques provide mechanisms for reasoning about systems at the architectural level. These techniques examine the structure and behavior of a system utilizing a high-level view. They are used to identify strengths and inadequacies in architectural designs.

In Section 2 of this chapter, the main concepts of software architecture are introduced. Section 3 discusses various types of software architectural evaluation techniques. Section 4 presents the details of a new process for software architectural evaluation. Section 5 describes several case studies where this process was used. Finally, Section 6 presents a summary and describes several opportunities for future work.

2. Software Architecture

Many definitions of software architecture exist; however, there is no universally accepted definition.[1] The lack of a standard definition often leads to confusion making a discussion of the definition of software architecture necessary before the topic of architectural evaluation can be introduced.

2.1 Sample Definitions

Although there are many definitions of software architecture, common themes and ideas run through these definitions. Many of the definitions include the concept of components or modules and the communication mechanisms among components. For example, Perry and Wolf [28] define software architecture by describing three classes of architectural elements: processing elements, data elements and connection elements. Data elements contain the information that is processed, used and manipulated by the processing elements. The connection elements are the parts of the system that connect the pieces of the system together. Bass, Clements, and Kazman [6] describe software architecture as the structure or structures of the system comprised components and externally visible properties and relationships among them. This definition implies that the internals of each component do not play a role in software architecture. Crispen and Stuckey [15] define software architecture as a partitioning strategy and a coordination strategy. The partitioning strategy defines how the system is divided (or composed) of components. The coordination strategy defines how the components of the system interface with each other. This definition for software architecture incorporates the ideas of constraints and rationale for how components are put together and how the communication should occur. Including the ideas of constraints and rationale is common in many definitions of software architecture. The Bass, Clements, and Kazman definition implies that behavior is part of the architecture. Clements and Kogut [14] include constraints and rationale in their definition of software architecture. Booch, Rumbaugh and Jacobson [10] also include constraints in their definition of software architecture. In several definitions, the concept of parameterized components that can be altered to form new applications is included. Bhansali [9] calls this type of architecture generic software architecture. The idea of a generic architecture seems to lead toward the concepts of architectural styles and design patterns as described in [20] and [19] respectively. Architectural styles and design patterns are important concepts in the architectural evaluation process that will be described later.

[1] The Software Engineering Institute maintains a list of definitions for software architecture and a related bibliography at http://www.sei.cmu.edu/architecture/definitions.html.

In some cases, software architecture is viewed from different levels or various perspectives. For example, Soni, Nord, and Hofmeister [31] describe four perspectives of software architecture: the conceptual architecture, the module interconnection architecture, the execution architecture, and the code architecture. The conceptual architecture describes the major design elements of the system and the relationships among them. The module interconnection architecture encompasses the functional decomposition of the system and its layers. The execution architecture describes the dynamic structure of the system. The code architecture describes how the source code of the system is organized. Kruchten [23] describes a $4 + 1$ (logical, process, physical, development, and use-case scenarios) view of software architecture. The views of the Kruchten model of software architecture are similar to those of Soni, Nord and Hofmeister. However, Kruchten includes use-case scenarios as an additional view of the software architecture.

2.2 Definitions Used in this Chapter

The definitions used in this chapter are based on common elements from the definitions of software architecture described in the previous section. Software architecture deals with the structure and interactions of a software system. The most basic building blocks of the structure of software architecture are components and the interrelationships among the components. In addition to structure, behavior is part of software architecture. Constraints and rules describe how the architectural components communicate with one another. The software architecture of a system may be viewed at different levels for different purposes.

Even at the conceptual or logical level, architectural components can be viewed at several different abstraction levels and vary based on the size of the system. At the highest level, the components are the subsystems, which in the case of a very large system, can be a complex system and can have subsystems of their own. Subsystems are often formed by lower level components, which are, in the case of an object-oriented system, collections of classes. In order for the architectural components to form a system, they must communicate with each other, creating interrelationships or connectors. The division of a system into smaller building blocks is based on the philosophy of "divide and conquer" which lets the implementer divide the problem into smaller and less complex pieces.

When viewed at the highest levels, a system's architecture is referred to as the macro-architecture of the software system. At lower levels of abstraction, it is referred to as micro-architecture. Architectural styles and design patterns are similar to what Bhansali [9] describes as generic forms of software architecture. Often architectural styles guide the structure and interactions of the system when describing

the software architecture of a system at the macro-architectural level. When describing the structure and/or interactions of a system at a micro-architectural level, often design patterns can be used.

2.2.1 Macro-Architecture—Architectural Styles

In Garlan and Shaw [20], several basic architectural styles are defined. One example is a client-server architectural style in which the application is divided into two major subsystems that communicate with each other. Other examples of architectural styles include layered and piped architectures. Each architectural style brings not only certain properties to the system, but also implicit and explicit implications and tradeoffs that translate into design rules and guidelines. The client-server architectural style, for example, divides the system into a "front-end client" and a "back-end server" allowing for a separation of concerns on a high level. This style results in a design guideline that restricts the communication between the client and the server to a common and narrow interface. The layered architecture has similar properties. It allows encapsulation of low-level functions and the establishment of an abstract machine that builds on the services lower layers offer. An implication of a layered architecture is, for example, that high level layers can only use services of lower level layers and not vice versa. Restricting communications to the layers directly below is another example of a rule that might follow from this choice in architectural style.

2.2.2 Micro-Architecture—Design Patterns

Design patterns methodically name, explain, and evaluate important and frequent designs in object-oriented systems. Design patterns "solve specific design problems and make object-oriented designs more flexible and elegant, and ultimately reusable. They help designers reuse successful designs by basing new designs on prior experience. A designer who is familiar with such patterns can apply them immediately to design problems without having to rediscover them" [19]. One example of a design pattern is the mediator that encapsulates how a set of objects interacts. Another example is the observer that typically is used to provide various views of the same subject data. The mediator and the observer are described and exemplified respectively in Sections 5.4 and 5.5. Design patterns are sometimes referred to as micro-architectural styles and contribute to an organized system structure. In each system, several design patterns can be used, and they can coexist with each other and with different architectural styles. By selecting a design pattern, certain architectural implications and tradeoffs follow. Some of these tradeoffs and implications are described at a high level in [19]; however, context-specific implications and tradeoffs may exist and result in implicit and explicit design rules and guidelines.

Typically, design patterns are discussed at the class level putting them into the category of micro-architecture. However, the concepts in some design patterns can be abstracted to the component level. To incorporate the ideas of design patterns into this higher-level view of software architecture, the class roles of a design pattern are abstracted to the component level. For example, in the Observer pattern, the role of the Subject class can be played by a component of classes representing the subject data to be observed by Observer components. Similarly, with the Mediator pattern, a set of classes (i.e., a component) may represent a Colleague in the pattern.

3. Software Architectural Evaluation

Software architectural evaluations are investigations into how a system is structured and behaves with the purpose of suggesting areas for improvement or understanding various aspects of a system (e.g., maintainability, reliability, or security). In many cases, a software architectural evaluation is performed before a system has been designed or implemented. Often, this type of architectural evaluation is performed to compare alternatives or to determine whether or not the architecture is complete or appropriate for the application. In other cases, a software architectural evaluation is performed after the system has been implemented. This type of architectural evaluation typically is performed to make certain that the actual implementation of a system matches the planned architectural design.

3.1 Pre-Implementation Software Architectural Evaluations

A software architectural evaluation that occurs prior to the implementation of a software system is referred to here as pre-implementation software architectural evaluation. Pre-implementation software architectural evaluation is used to evaluate one or more software architectural candidates or to evaluate various properties of an architecture. Pre-implementation software architectural evaluations, as described by Abowd et al. [1] and Avritzer and Weyuker [3], can be based on the description of the software and other sources of information such as interviews with the designers of the software architecture. With these methodologies, scenarios are often used to determine the adequacy of the software architecture. A scenario is a series of steps that describes the use or a modification to the system from the point of view of different stakeholders. Scenarios are used to highlight potential weaknesses and strengths of the architecture and are useful for concretizing requirements of the system and understanding non-functional requirements. An example of a scenario is: when a data exception occurs, all users should be notified by e-mail immediately. This scenario indicates that the system should be reliable. Scenarios like these should be

taken into account when evaluating the architectures so that the best solutions are chosen. The methodology proposed by Yacoub and Ammar [34] analyses architectures from a risk perspective early in the life cycle to assess whether the system will have the desired qualities or quality attributes (e.g., performance, reliability, and maintainability). Bengtsson and Bosch's architectural level prediction of software maintenance is another example of the same kind of analysis [8]. The approach suggested by de Bruijn and van Vliet [11] is based on generating a basic architecture that is immediately evaluated to identify problems. The cycle of generating and evaluating the software architecture is repeated. This iterative approach continues until the architecture fulfills the software requirements. The Architecture-Level Modifiability Analysis (ALMA) model by Lassing et al. [24] emphasizes that a goal should govern how the evaluation of the software architecture is carried out. ALMA makes a distinction between three different goals for evaluation: risk assessment, maintenance cost prediction, and software architecture comparison. Architectural evaluations with the goal of risk assessment focus on finding types of changes for which the system is inflexible. For maintenance cost prediction, software architectural evaluations estimate the cost of maintenance effort for the system during a given period. When software architecture comparison is the goal of the evaluation, two or more candidate software architectures are compared to find the most appropriate one. Additional software architectural evaluation approaches are described in Dobrica and Niemela [17].

3.2 Implementation-Oriented Software Architectural Evaluations

Implementation-oriented software architectural evaluation is a term used to describe the evaluation of software architecture after a version of the system has been implemented. Since this type of software architectural evaluation is performed after a version of the system exists, it can utilize data measured from the actual source code and associated documentation. Implementation-oriented software architectural evaluations can be used for similar goals to pre-implementation software architectural evaluations. For example, the source code and associated documentation can be used to reconstruct the *actual software architecture* in order to compare it to the *planned or conceptual software architecture*. Recovering the actual architecture of an implemented system can be used for risk assessment and maintenance cost prediction as well. The analysis of the actual software architecture can be used to evaluate whether the implemented software architecture fulfills the planned software architecture and associated goals, rules and guidelines.

A literature survey conducted by Hochstein and Lindvall [21] noted that whereas there are many pre-implementation software architectural evaluation approaches available, there are not many implementation-oriented approaches described in the literature. An early example of an implementation-oriented approach is Schwanke's re-engineering tool [29]. The tool uses a concept of similarity based on Parnas' information hiding principle; but is not designed specifically for object-oriented systems. Murphy et al.'s work with reflexion models [27] is another example of this type of software architectural evaluation. With reflexion models, the designers of the system are interviewed to describe the conceptual view of the implemented system. Next, the actual architecture is recovered to determine if the designer's conceptual view matches the implementation.

Implementation-oriented software architectural evaluation can be combined with pre-implementation software architectural evaluation to achieve different goals. Before implementation occurs, software architectural evaluation is used to assure that certain goals are achieved based on the architectural decisions made. However, in many cases, the implementation of a system differs from the architectural design. That is, the implementation often does not match the planned design. Software systems are subject to changes over their lifetime. When time is spent evaluating architecture to meet certain criteria and to achieve specific quality attributes prior to system implementation, there is a need to assure that each new version of the system indeed matches the planned architecture to harvest the benefits of the initial investment in architectural evaluation. Implementation-oriented software architectural evaluations can be used to make certain that the actual architecture stays on the intended track.

An example of a combination of approaches that would be successful is the combination of the Architecture Trade-off Analysis Method (ATAM) [13] and an implementation-oriented approach like the one described later. The ATAM method uses brainstorming sessions to perform a tradeoff analysis of architectural decisions based on their impact on different quality attributes that are important for different classes of stakeholders. The ATAM method is valuable in an early phase of the development when the design is being created. Once the architecture is evaluated, design decisions are made and these decisions are expected to be implemented in the system. An implementation-oriented approach such as the one described later could be used after a version of the system has been developed to (i) make sure the architecture maintains the properties desired to reach a certain quality attribute and (ii) make sure the implementation conforms to the planned architectural design. Every version of the system can be assessed and evaluated in order to track the architecture and keep it aligned with the design goals.

4. A Process for Implementation-Oriented Software Architectural Evaluation

The process defined in this chapter is one that evolved over several iterations of software architecture evaluations. This software architectural evaluation process utilizes the macro-architecture view of a system. The process analyzes the components and their external interfaces with each other. This process is conducted after an implementation of the system exists.

The process defined here is not a substitute for alternate architectural evaluation processes that are typically conducted before a system is implemented. Rather, this process may be used in conjunction with other architectural evaluation techniques to ensure that the system is being implemented in agreement with the recommendations resulting from other forms of architectural evaluation. For example, if an architectural evaluation performed before the system is designed results in a suggestion of a recommended architectural style, the architectural evaluation process described in this section can be used to confirm that the suggested architectural style has been implemented correctly in the actual system.

The process was designed to be relatively quick and used repeatedly to keep the actual implementation of the software architecture consistent with the planned architectural design. It does not assume that documentation of the architectural design exists prior to the evaluation of the system. However, if documentation does not exist prior to the evaluation, the process requires input from the developers of the system to recover the planned architectural design of the system. The architectural evaluation process involves members of the development team and ideally, a separate analysis team. One of the main objectives of this process is to provide useful feedback to the development team without extensively disrupting their daily activities. The members of the development team are used sparingly in this process to confirm the findings of the analysis team.

4.1 Background

One of Fraunhofer Center for Experimental Software Engineering, Maryland's (FC-MD) primary assets is a software system, the Visual Query Interface (VQI) [30], which manages experience supporting the Experience Factory approach for organizational learning [4]. In 1999, the system had evolved into a difficult state. Both current and potential users of the system wanted more functionality, but it was clear that the current architecture of the system could not withstand additional changes. Each change to the system became increasingly difficult to make. The effort of adding new functionality widely exceeded expectations and it had become apparent that the

system had decayed [18] to a point its maintenance had become a problem; making it clear that the system need to be restructured.

After the investment in restructuring the system was made, it was important to FC-MD to confirm that the structure of the system had improved and that it had been implemented as planned. The software architectural evaluation process described in the next section was developed to assist with this task.

4.2 Definition of the Process Steps

The architectural evaluation process consists of seven process steps. As mentioned previously, one of the main objectives of this process is to provide quick and useful feedback to the developers of a project. To minimize negative schedule effects, it is envisioned that an analysis team, separate from the development team, performs the majority of the work for the evaluation process. The following description of each process step contains information regarding the roles of the development and analysis teams. It is envisioned that this process can be tailored for different contexts using different metrics and representations of the architecture. This subsection provides the basic outline of the steps and describes the roles for the analysis team and the developers. The case studies described in the next section give a more detailed view of how the process could be applied.

Step 1. Select a Perspective for Evaluation

Architectural evaluations may be conducted with different goals in mind and from many different perspectives. For example, a system might be evaluated to determine whether or not the system implements the specified functional requirements or to determine if it fulfills the non-functional requirements, i.e., the system qualities or quality attributes. Other examples of perspectives are evaluation for security, reliability, performance, and maintainability. Selecting a perspective is important for identifying appropriate goals and measurements for the evaluation process. The GQM technique [5] is used in this step to define goal-oriented metrics based on questions that need to be answered to determine if goals have been achieved. For example, if the perspective chosen is maintainability, goals based on attributes of maintainability such as coupling and cohesion might be identified.

The analysis team performs this step with the help of the development team. The development team provides input on the perspective. The analysis team creates the goals and defines the metrics following the GQM technique. The development team provides feedback to the analysis team on the goals and metrics.

Step 2. Define Planned Architecture and Guidelines

Once a perspective has been chosen and goals have been identified and elaborated with the GQM technique, the planned architecture of the system is identified and guidelines with associated metrics are defined. These architectural guidelines are used to validate that the architecture possesses desired properties. Often, these guidelines translate into quantitative metrics. For example, if evaluating a system from the perspective of maintainability, guidelines related to coupling might be established. Sample guidelines based on coupling might include the following: coupling between the components should be low; and the extent of the coupling between components should be low. Quantitative metrics measuring coupling between the components and the extent of the coupling are derived from these guidelines. In addition to defining guidelines and metrics based on the perspective of the architectural evaluation, some guidelines and metrics are defined based on the architectural styles and the design patterns chosen for the system.

The analysis team works with one or two representatives of the development team (and/or uses documentation) to identify the planned, high-level architecture of the system. The planned (or ideal/intended) architecture is defined by architectural requirements, by implicit and explicit architectural guidelines and design rules and implications stemming from the use of architectural styles and design patterns. The analysis team needs to recover the different aspects of the planned architecture and create a model of it that will guide the evaluation. Once the high level architecture of the system has been defined, the analysis team uses it to derive the implications, tradeoffs, guidelines, and design rules that result. The analysis team needs to select and customize the guidelines and metrics for the specific context. The selected set of metrics must capture the properties that the team finds most important while, at the same time, being cost-efficient to collect and analyze. As the analysis team learns more about the planned architecture, these guidelines and metrics are commonly iterated and updated during this step.

Step 3. Recover Actual Architecture

The actual architecture is the high-level structure of the implemented system, its architectural components and their interrelationships, as well as its architectural styles and design patterns. Studying the implementation of the system, which is, to a large extent an abstraction obtained from the source code, represents the actual architecture. It should be noted that this step is not the same as source code analysis, but it is used to identify the static architectural components of the actual system. To perform this step efficiently, the analysis team relies on a set of automated or partially automated tools that help them with this task. In many cases, the tools have to be

defined based on programming language, the measurements that are to be collected, and other factors in the development environment.

Identifying the contents of a component is often one of the key complications involved with the recovery of the high-level architecture of an implemented system. In some cases, programming language features can be used to reduce some of the difficulties associated with this task. With Java, for example, the analysis team can use packages as a way of determining the contents of the system's components. However, not all Java developers use packages and even when packages are used, there is not always a one-to-one correspondence between the packages and the high-level components of the system.

Identifying architectural styles and design patterns is another complication that arises with the recovery of the actual architecture. Architectural styles are not always easy to detect in the actual implementation of a system. Design patterns can be implemented in different ways and can be difficult to detect.

As part of this step of the process, the design team works with one or two members of the development team to partition the files containing the actual implementation of the system into their appropriate components. Then, the analysis team extracts relevant information and computes metrics from the component files to obtain the actual architecture of the system.

Step 4. Compare Actual to Planned to Identify Architectural Deviations

Architectural deviations are differences between the planned architecture and the actual implemented version of the architecture. These architectural violations are identified by comparing the planned architectural design defined in step two to the abstraction of the actual architecture obtained in step three. Deviations can be missing or extra components, missing or extra connections between components, violations of architectural guidelines, or values of metrics that exceed or do not match a certain expected value.

The analysis team compiles a list of these violations and notes the circumstances under which the violation was detected and the reason the team suspects it is a violation. If necessary, the analysis team conducts a more detailed analysis of the deviation in order to determine its possible cause and degree of severity. The deviations are categorized and patterns of violations are identified.

Step 5. Verify Violations

Once the analysis team has composed and characterized the list of architectural violations, the list is verified during a discussion with one or two members of the

development team. This step is taken for several reasons. First, it helps to ensure that the analysis team has not incorrectly identified any violations by a misunderstanding of how the system was implemented. Secondly, it gives the development team feedback on how well the actual implementation matches the planned architecture and exposes the general types of deviations that have occurred. Additionally, the analysis team gathers more information on how and why the violations have occurred.

Step 6. Suggest Changes to Planned and/or Actual Architectures

Based on the results from the previous step, the analysis team formulates high-level change recommendations that would remove the deviations from the system. Sometimes, the deviations result in requests for source code changes. In some cases, the requests are related to changes in the planned architecture or guidelines. It should be noted that it is not the task of the analysis team to design or implement the change requests. Rather, this step is a way for the analysis team to contribute to the improvement of the system in a constructive way by giving feedback to the development team.

Step 7. Repeat Steps 4–6 After Changes Have Been Implemented

The analysis team discusses the change requests with the development team. It is the role of the development team to decide which changes to implement and how the changes will be implemented. Once the changes have been implemented, it is important to verify that actual architecture complies with the planned one. To verify that the planned and actual architectures are in alignment, the steps of identifying the actual architecture and any architectural deviations are repeated. This verification is done to make sure that the changes have been implemented correctly and that no new violations have been introduced into the system. The steps of the process are shown in Fig. 1.

4.3 Use of the Process

The process for software architectural evaluation is designed to be efficient and conducted at various points over the lifetime of a software system. It is expected that one iteration of the evaluation process will be run after a new feature or set of features is implemented. Another important objective in the design of this process was to provide valuable feedback to the development team with minimal disruption to the team's usual activities. The process was designed to fit seamlessly into an existing

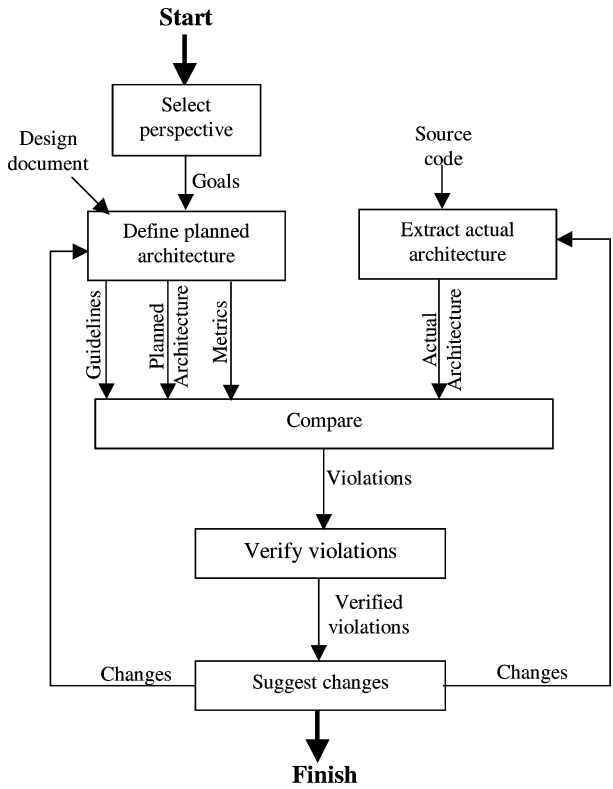


FIG. 1. A diagram of the software architectural evaluation process steps.

development environment. The analysis team should perform the bulk of the work needed to conduct the evaluation. Members of the development team should be used to verify the findings of the analysis team in both the definition and recovery of the architecture and implement the recommended changes.

5. Case Studies

The software architectural evaluation process was created due to a maintainability problem experienced with a project developed by FC-MD. The architectural evaluation process has been used to analyze several iterations of a FC-MD software tool (VQI) and with one commercial product prior to its release. The commercial prod-

uct is a proprietary system that incorporates the design, simulation and analysis of CAD diagrams. In both of these case studies, the perspective chosen for the evaluation was maintainability, and the metrics and representation of the architecture used in the evaluation were identical. The case studies are presented here in the order in which they were performed. The first case study summarizes the results of several iterations through the evaluation process on the VQI system. The second case study gives a more in-depth description of a single iteration through the evaluation process on the commercial product.

5.1 Perspective for Evaluation—Maintainability

With the VQI system, the main goal of the initial re-engineering activity was to create a system that was easier to maintain than the previous version that had degenerated into an inflexible state. The initial architectural evaluation was conducted to recover the actual architecture of the existing decayed system and to verify that the re-engineered system conformed to the new design plans. For each of the following versions, the evaluations were conducted to verify that the changes made to the system did not deviate from the planned architecture and design guidelines.

With the proprietary commercial product, the architectural evaluation was conducted before the product was released for beta testing. The main objective for this evaluation was to verify that the system had been implemented according to the plan and to identify potentially fault-prone components.

In the case study evaluations, coupling was the main characteristic considered for determining maintainability. In both the VQI and the proprietary commercial system, low coupling between components was considered to be desirable. However, based on component characteristics such as the design patterns chosen, several components were expected to have higher coupling than other components of the system. For example, a distinction is made between components that are library-oriented and those that are function-oriented. A library-oriented component is one that is made up of related data structures and routines that are expected to be used by many other components. A function-oriented component is one that implements a specific functional requirement of an application. Given these definitions, high coupling from non-library components to library-oriented components is expected while coupling from library-oriented components to non-library components is undesirable, and the coupling among function-oriented components should be minimized.

5.2 Definition of Metrics

The metrics used in the evaluations were based on the notion of static couplings between modules and classes detected in the source code. Essentially, a general de-

sign goal is to reduce the amount of inter-module coupling without increasing the amount of intra-module coupling. To measure the degree of inter-module coupling, two metrics were defined: coupling between modules (CBM) and coupling between module classes (CBMC). A complete description of these metrics and a discussion of how they were derived can be found in [25] and [26].

For the definitions of CBM and CBMC, static coupling is considered. A coupling between two classes occurs when one class uses another class in its implementation. Coupling occurs when the source code of one class includes the definition of another class (e.g., importing in Java and including in C++), includes a declaration of a variable or parameter of another class type, or accesses a data member or method of another class. When the two classes involved in the coupling belong to different components, the coupling represents a coupling between components (or modules).

The CBM metric was defined to capture the degree of coupling between components in a system at a coarse level. This metric is similar to the coupling between objects (CBO) metric as defined in [12]; however, instead of counting coupling at a class level, CBM is calculated at the component level. Components with high CBM values are potential areas for maintenance problems or signs of system degeneration. While having a few components with high CBM values may not represent a problem and may be desirable, an architecture that contains many components with high CBM values indicates a poorly structured system. In such a system, the components are highly interrelated, and a change to one component likely will affect many components of the system.

To capture the level of coupling at a finer level, the metric CBMC was defined. While CBM calculates coupling at the component level, this metric is used to measure the number of classes involved in the coupling between modules. As with the CBM metric, a high CBMC value indicates modules that are interdependent and may represent maintenance problems or system degeneration. The CBMC value gives an indication of the "width" of the interface between the two components. If a component has a high CBM value and a high CBMC value, changes to the component most likely will be difficult due to its interrelationships with many classes in many components.

To illustrate how these metrics are calculated, consider the example of the high-level architecture of a system given in Fig. 2.

When calculating the CBM metric, the coupling among the classes of a single component is ignored. For example, the couplings between classes Y and X and Y and Z within component A are ignored in the calculation of the CBM metric value. Additionally, the direction of the coupling is ignored. Once the intra-module couplings and the direction of the couplings are ignored, the architecture of the system can be viewed as shown in Fig. 3 and the CBM values for each component can be calculated.

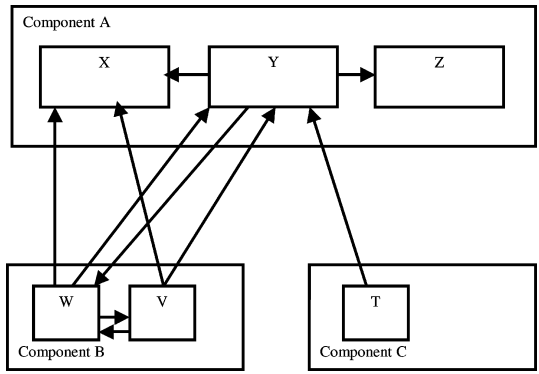


FIG. 2. Example diagram of a high level architecture of a system.

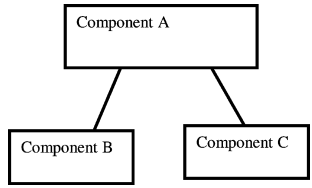


FIG. 3. View of example system with intra-module couplings and the direction of the couplings removed.

In this case, the CBM value for component A is 2 and is denoted by $\mathrm{CBM}(A) = 2$. Likewise, the $\mathrm{CBM}(B) = 1$ and the $\mathrm{CBM}(C) = 1$.

To calculate the CBMC values for each component, the intra-module couplings are eliminated as shown in Fig. 4. Again, the direction of the arrows is ignored, but the number of classes involved is not. In this case, the CBMC value for component A is 5 and denoted by $\mathrm{CBMC}(A) = 5$. Similarly, the $\mathrm{CBMC}(B) = 4$ and the $\mathrm{CBMC}(C) = 2$.

Retaining the direction of the coupling arrows provides another useful view of the coupling between modules and the degree of the coupling between two components along the direction of the coupling. For this view, the classes are not drawn in the diagram, but each directed arrow is labeled with the number of classes involved in the coupling along the direction of the arrow. The example system with labeled directed arrows is depicted in Fig. 5. This view is useful for determining the width of the interface between two components. In the example system, for instance, the coupling from component C to component A is narrow since only one class within component A is being accessed by component C. By looking at this view of the ar-

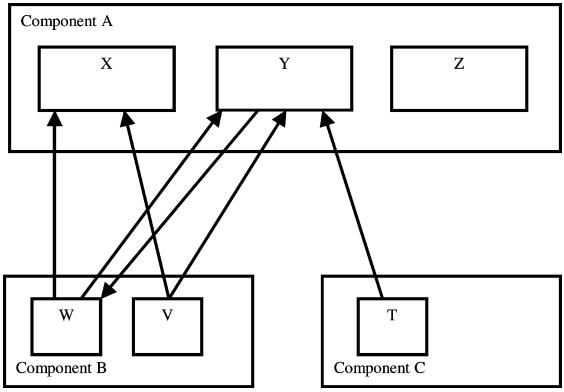


FIG. 4. Example system with intra-module couplings eliminated.

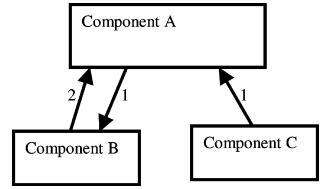


FIG. 5. Example system with direction of coupling included. The label represents the number of classes being accessed in the component at the endpoint of the arrow.

chitecture, components with wide interfaces (i.e., the arrows with large labels) can be identified easily.

5.3 Representing the Architecture

In both case studies, the same representation was used for the architecture of the systems. Boxes are used to represent high-level components of the system. Each high-level component box represents a group of Java classes in these cases. Because of the metrics used for these studies, arrows are used to represent the coupling relationships among the components of the system.

For the planned architecture, the boxes and directional arrows are defined based on the architectural styles, design patterns, and guidelines identified. For example, if the system used a layered architecture, the representation of the planned architecture might include component boxes for each layer with arrows going from one layer to the next.

Figure 6 shows a simple example of how the guidelines for a layered architecture might be translated into a representation of the planned architecture. In this example, there are three components (i.e., TopLayer, MiddleLayer, and BottomLayer). The classes within each layer are expected to communicate only with those layers that are immediately adjacent to the layer. For the case studies presented here, this planned architectural diagram was translated into a text file. The text file representation of the planned architecture shown in Fig. 6 is shown in Table I. The first line of the text file gives a name to the diagram. The subsequent lines define a component and the components that are expected to communicate with the defined component.

The representation of the actual architecture used in the case studies is derived from the source code. A parser is used to extract static coupling information (as described in the previous subsection) from the source code of the system. The output of the parser is a comma-separated file that is used to create the diagrams representing the actual architecture of the system. Each line contains coupling information including the From Subsystem, From Component, From Class, To Subsystem, To Component, To Class. An example of a subset of the output from the parser is shown in Table II.

To create the diagrams of the actual architecture from this file, the From Component and the To Component fields are used. A box is drawn for each component. When the From Component field differs from the To Component field, an arrow

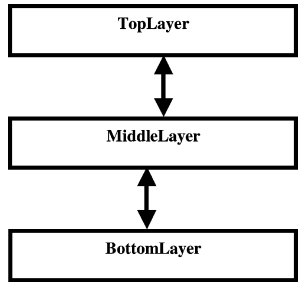


FIG. 6. Example representation of a planned architecture.

TABLE I
A TEXT FILE REPRESENTATION OF THE PLANNED ARCHITECTURE SHOWN IN FIG. 6

```
Layered;
TopLayer:MiddleLayer;
MiddleLayer:TopLayer,BottomLayer;
BottomLayer:MiddleLayer;
```

TABLE II
SAMPLE OUTPUT OF THE ACTUAL ARCHITECTURE FROM THE PARSER

From Subsystem	From Component	From Class	To Subsystem	To Component	To Class
Client	Cache	CacheObject	Client	Mediator	Mediator
Client	Connector	ServletProxy	Client	Mediator	Mediator
Client	Editor	EditorObject	Client	Mediator	MediatorObject
Client	Editor	EditorObject	Client	Mediator	Mediator
Client	Editor	EditorObject	Client	Editor	PackageTypeSetup
Client	Editor	EditorObject	Client	Editor	Hyperwave
Client	Editor	PackageTypeSetup	Client	Editor	EditorObject
Client	Logger	EMSLogger	Client	Mediator	MediatorObject
Client	Logger	EMSLogger	Client	Mediator	Mediator

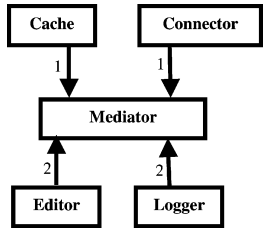


FIG. 7. Diagram of actual architecture based on partial subset of data shown in Table II.

is drawn from the box representing the From Component to the box representing the To Component. The number labels for the edges are computed by counting the number of distinct classes in the To Component reached from the From Component. The subset of information shown in Table II would lead to the diagram of the actual architecture shown in Fig. 7.

5.4 Case Study 1—the VQI

The implementation-oriented software architectural evaluation process was originally developed and used to evaluate several versions of the FC-MD system as described previously. As the VQI system was restructured and evolved, architectural evaluations were conducted to confirm that the actual system changes had been implemented as planned.

TABLE III
SIZE CHARACTERISTICS OF VQI

Version	Size (LOC including comments)
VQI1	17,522
VQI2	24,269
VQI3	26,915
VQI4	27,072

5.4.1 Background

The architectural evaluation process was used on three iterations of the VQI system. The versions of the VQI system are numbered VQI1–VQI4 indicating the order in which they were developed. Size characteristics for each version are presented in Table III.

Initially, the VQI was developed by two people and amounted to about 17K lines of commented source code (LOC) written in Java. VQI2 represents a significant architectural change from VQI1 due to maintainability problems with the previous version. The VQI2 system contains over 24K lines of commented Java source code developed by four local developers plus a team of three developers in Brazil.

VQI3 represents the work done by one developer to implement a specific task. VQI3 also represents a major change to the system in terms of increased usability and new capabilities and was released to end-users. Examples of changes in version three include converting VQI to run as both a desktop and a web application; connecting and drawing data from several different data sources instead of a single data source; and adding several advanced graphical and statistical analysis features.

VQI4 is a version of the system that is a result of an effort to eliminate the set of detected architectural violations in VQI3. The difference between VQI3 and VQI4 is relatively limited. No new features were added to this version of the system. VQI4 represents the system after changes were made specifically to correct architectural deviations.

5.4.2 Planned Architecture

5.4.2.1 Architectural Style. The overall architectural style of the VQI2–VQI4 versions is an Internet-based and client-server system with most of the functionality and maintenance efforts residing in the client. The server is responsible only for handling requests from the client. When the client requests data from the server, the server connects to a database, retrieves the requested data, processes it and sends the data back to the client. The client consists of the majority of the code that im-

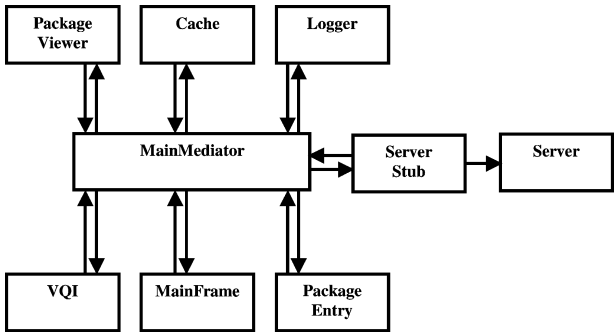


FIG. 8. A diagram of the planned architectural design for VQI2.

plements most of the functionality of VQI2–VQI4. Figure 8 shows a diagram of the planned architecture for VQI2.

5.4.2.2 Design Pattern. The core of the architecture for client of VQI2–VQI4 is based on the mediator design pattern [19].

The mediator is one of the behavioral patterns described in [19]. The pattern encapsulates how a set of objects interact, providing loose coupling among the objects where the objects do not reference each other explicitly. The pattern consists of a Mediator object and Colleague objects. The Colleague objects are not aware of each other and only share information with each other through the Mediator object. The Mediator object is responsible for coordinating the communications among all of its Colleagues.

In the client side of VQI2–VQI4, the mediator design pattern is abstracted to a component level. The Mediator is a component of classes and interfaces related to controlling the communication among Colleague components. The Colleague components implement different functionalities in the client and interact with each other only through the classes of the Mediator component. In each of the Colleague components, only one class is supposed to interact with the Mediator component directly. The Server Stub component is the Colleague that handles the communication with the Server. Hence, it has interactions with the Mediator component and the Server component.

5.4.3 Guidelines

A set of design guidelines for the VQI was derived based on the properties of intra-module coupling, the client-server architectural style and the mediator design

pattern. The following guidelines represent sample design guidelines that are specific to the VQI system:

- [DG2] The Server should contain no references to the Client (since the server does not initiate communication with the clients).
- [DG6] The Mediator should be coupled with exactly one class per component and vice versa.
- DG2 is an example of an architectural style guideline, based on the client-server architectural style. DG6 is an example of a design pattern guideline reflecting how the mediator design pattern should appear in the implementation. The details of the metrics defined during these evaluations and a complete set of guidelines are described in [6] and [7].

5.4.4 Results from the VQI2 Evaluation

The architectural evaluation of the changes implemented in VQI2 was the first iteration through the evaluation process. During this iteration, many of the guidelines, metrics and tools used in future iterations were developed. Since VQI1 was a research prototype with no official design document, the existing architecture had to be recovered and a planned design had to be created before the implementation of VQI2. When the time came for the analysis team to perform the architectural evaluation, the high-level planned design for VQI2 was readily available. Having this high-level design document allowed for a quicker evaluation. Since members of the development team were closely involved with the planned architectural design of VQI2, the expectations were to find no deviations between the planned and actual architectures.

During this evaluation, a tool was developed to extract from the source code the interrelationships among components. The tool produced tables that were imported into Microsoft Excel spreadsheets and diagrams of the actual architecture were constructed manually. Figure 8 shows the diagram of the planned design for VQI2. Figure 9 shows the actual design for VQI2 that was constructed manually based on the spreadsheets generated by the tool. These diagrams were used to identify violations of the architectural design guidelines. The first iteration through the evaluation process took several weeks to conduct because new metrics and guidelines based on the architectural style and design patterns of the VQI were developed.

Although the expectations were to find no deviations, the architectural evaluation surprisingly uncovered four architectural violations during the first iteration. Three of the violations can be seen in comparison diagram (Fig. 9). The fourth violation was a metric guideline violation. The violation from the PackageEntry component to the VQI component was a design pattern guideline violation and was considered to

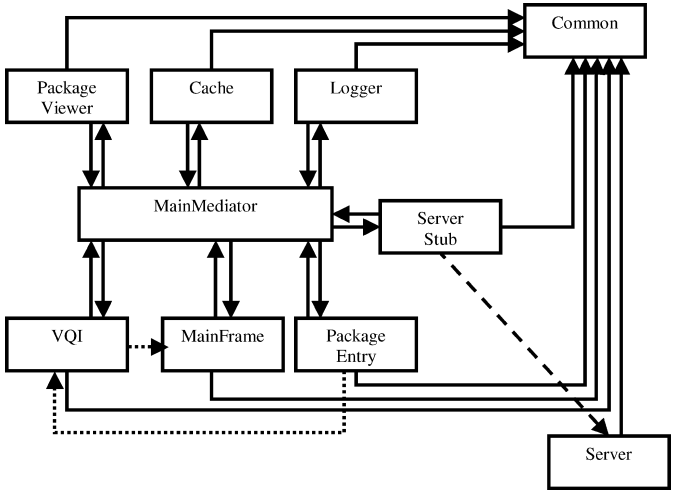


FIG. 9. A diagram of the actual architecture of VQI2. Extra coupling violations are shown with dotted lines. Missing coupling violations are shown with dashed lines.

be serious. The other violations were considered to be minor; however, one resulted in a change to the original metric guidelines.

From this iteration of the evaluation process, several resources were created for future versions of the VQI and for future iterations through the evaluation process. These resources include a set of metrics, architectural guidelines, and a diagram of the planned architecture. In addition to the VQI-specific resources, the first iteration of the process also resulted in the creation of a tool to extract the static coupling information from the source code. Although the first evaluation took several weeks to perform, the metrics, guidelines, and tool support enabled faster evaluations of new versions of the VQI system.

This first iteration through the evaluation process provided several lessons. Having members of the design team participate in the development does not necessarily prevent the occurrence of serious architectural violations. In this case, the main objective of the development was to re-engineer the system into a more maintainable state. Developers actively participated in the re-design of the architecture of the system. Architectural mismatches were not expected in the actual implementation, yet they still occurred.

The experiences from this iteration indicate that implementing design patterns may not be entirely straightforward. The goals of the architectural design and the implementation details of the design pattern were explained to the developers. Still, design pattern violations appeared in the implementation. Although it is believed

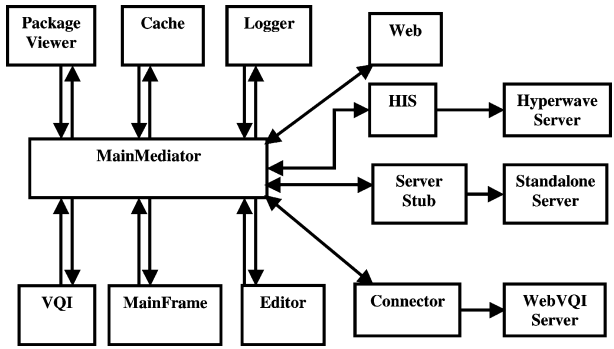


FIG. 10. A diagram of the planned architectural design for VQI3 and VQI4.

that the mediator design pattern has and will continue to make the VQI system easier to maintain, when new developers are introduced to the concept, understanding the implementation details of the design pattern has a learning curve.

5.4.5 Results from the VQI3 Evaluation

The evaluation of VQI3 took place after a visiting student had implemented a new requirement and another student had implemented several additional changes. These changes involved several adjustments to the high level architectural design of the VQI system. The planned architectural design for VQI3 is shown in Fig. 10. The developers that made the changes for VQI3 were not the same as those involved in the development of VQI2. This iteration of the evaluation made use of many of the same architectural guidelines and metrics that were used during the first iteration of the evaluation. However, instead of using the tool developed for the first iteration immediately, a "whiteboard" discussion with the developer was used to reconstruct the actual architecture of the system. During the course of this discussion, three design pattern violations were uncovered.

After the discussion with the developer, the static coupling tool developed during the previous iteration was used on the source code to confirm the results and completeness of the "whiteboard" discussion. A diagram showing the structural violations detected by the tool is shown in Fig. 11. The tool uncovered 15 violations of architectural guidelines. Ten of the 15 violations were metric guideline violations. Nine of the violations (including some that were also metric violations) were design pattern violations. One violation was a violation of a general architectural guideline.

Once again, a misunderstanding of the design pattern implementation details was the source of many of the uncovered guideline violations. During this iteration, several new components were added to the system. However, these components were

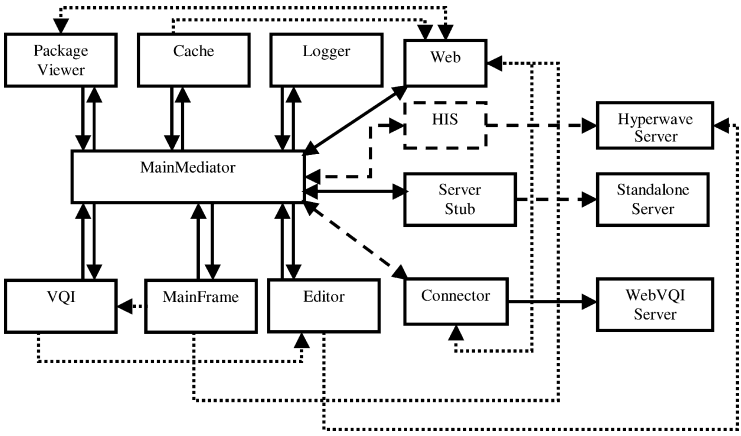


FIG. 11. A diagram of the actual architectural design of VQI3. Extra coupling violations are shown with dotted lines. Missing coupling violations are shown with dashed lines.

added to the system incorrectly. As with the previous iteration, the new components were not supposed to communicate with any other components directly. The communications with Colleague components were supposed to be coordinated through the MainMediator component. However, these new components bypassed the MainMediator component and accessed other Colleague components directly.

Because many guidelines for the VQI system and a tool to extract static couplings were produced during the first iteration of the evaluation process, the evaluation of VQI3 took only a few days. A few hours were spent with the developer in the "whiteboard" discussion. After running the tool on the source code, it took another day to investigate and confirm the violations.

During this iteration of the evaluation process, several new lessons about the process were learned. Once the high-level design and guidelines for a system are defined, evaluating new versions of the system becomes more efficient. This iteration through the evaluation process also demonstrated that while discussing the architecture with the developer is useful in uncovering some architectural problems, these discussions do not find all of the architectural problems that exist in the actual implementation of the system. Using tools to support the evaluation process is necessary.

5.4.6 Results from the VQI4 Evaluation

Since the evaluation of VQI4 was performed after the visiting student had made revisions to fix the violations uncovered in VQI3, the high-level planned design did

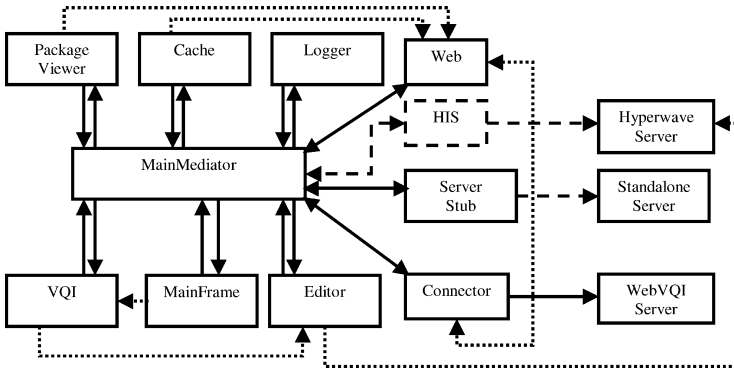


FIG. 12. A diagram of the actual architectural design of VQI4. Extra coupling violations are shown with dotted lines. Missing coupling violations are shown with dashed lines.

not change between versions three and four of the VQI. (See Fig. 10.) There were other minor modifications made to the source code, but nothing that affected the structure of the high-level design. The metric and design guidelines from the previous versions were also applicable to this version of the system. For this evaluation, the tool to extract the interrelationships from the source code was used to obtain an abstraction of the actual architecture.

The architectural evaluation of VQI4 found 12 architectural guideline violations. With VQI4, four existing violations were removed from the system while one new violation was added. The newly added violation was a minor violation. The violations that were removed were design pattern violations. A diagram showing the structural violations is shown in Fig. 12.

This iteration of the evaluation process only took a few hours to perform. It is believed that this iteration reflects the amount of time it would take to evaluate a typical system once the metric and architectural guidelines have been established. However, another factor affecting the length of time needed for an evaluation is the number of violations detected. When more violations are uncovered, more effort is required to investigate and suggest solutions to eliminate the violations.

5.4.7 *Summary of Results for VQI Evaluations*

Throughout the three iterations of the architectural evaluation process on the VQI system, 20 metric and design goal violations were identified. Many of the violations were ones that were considered to be serious violations that would threaten the maintainability of the system. A summary of the results from each iteration of the evaluation process on the VQI is given in Table IV.

TABLE IV
SUMMARY OF EVALUATION RESULTS

Evaluation version	Number of violations	Amount of effort
VQI2	4	A few weeks
VQI3	15	A few days
VQI4	12	A few hours

The majority of the violations found during the evaluation were problems related to the mediator design pattern. When new developers added components to the system, they did not connect them properly. Other less serious violations included classes being placed in the wrong package and importing classes that were not used. Although these violations are not as serious as a design pattern violation, they could still lead to confusion as the system develops. More information about the software architectural evaluations for VQI3 and VQI4 can be found in [32].

Although the architectural evaluation process yielded good results for the VQI system, there were several issues that needed to be addressed to determine the feasibility and applicability to other systems. The architectural evaluation process had been applied to the VQI system only. The VQI system is relatively small and had been designed and developed by members of the analysis team. Although a tool had been developed to support the process, some steps of the process were performed manually and sometimes resulted in erroneous diagrams. Additional tool support was necessary to make the process easier to apply.

5.5 Case Study 2—Proprietary Design, Simulation and Analysis Tool

To address some of the issues associated with the first set of evaluations, an additional architectural evaluation was conducted. For this evaluation, additional tool support was developed. The object of this case study was a system that was not known to the evaluators. This case study was conducted to determine if the process could be applied to a larger system that was unknown to the evaluators. Like the VQI, the perspective of this evaluation was maintainability. The metrics used in this study were the same as those used in the VQI evaluations.

5.5.1 Background

The subject of this case study was a proprietary computer aided design, simulation and analysis application. Because of the proprietary nature of the system, many of the details of the system must be withheld.

The system had not yet been released when the architectural evaluation was conducted. The system is written in Java and at the time of the evaluation, the system consisted of nearly 80 KLOC. The system was developed mainly by two developers. These two developers worked in two different geographic locations and met weekly to discuss the progress of the product. The system was developed with an extreme programming-like process. The requirements planning and testing practices of extreme programming (XP) [7] were the main practices of XP used in this development. As such, the developers did not have a true design document from which the code was developed, although the developers were expected to follow several design patterns and guidelines. As the system neared release and new programmers were hired to begin work on the development, the company wanted to conduct the architectural evaluation to be sure that the system was being implemented according to the guidelines and to communicate these architectural guidelines and design patterns to the new developers.

5.5.2 Planned Architecture

As mentioned previously, although there was no design document for the system, there were several architectural guidelines based on the architectural style and design patterns in place for the development of the system.

5.5.2.1 Architectural Style. The architectural style of the system under evaluation is a plug-in architecture. As with many aspects of software architecture, there is no universally accepted definition for the term plug-in architecture; thus, it is necessary to explain the definition used in this context. The main objective behind a plug-in architecture is to facilitate the extension and modification of a system's capabilities. To allow for easy adjustments and enhancements to the system without major changes to every component, coupling between the system components is restricted with this architectural style. The plug-in architecture is broken into three major component types: plug-in components, a plug-in manager component, and a component housing the requesting process.

Plug-in: Plug-in components are used to exchange data and control to and from a requesting process. The plug-in component provides specific services to a requesting process via a plug-in manager component using a well-defined interface. As long as a plug-in component implements the well-defined interface, the plug-in manager component will be able to detect and communicate with it.

Plug-in Manager Component: The plug-in manager component coordinates interactions between plug-in components and a requesting process. When a requesting process requires a service from a plug-in, it sends a request to the plug-in manager component, which in turn, communicates with the plug-in to execute the service.

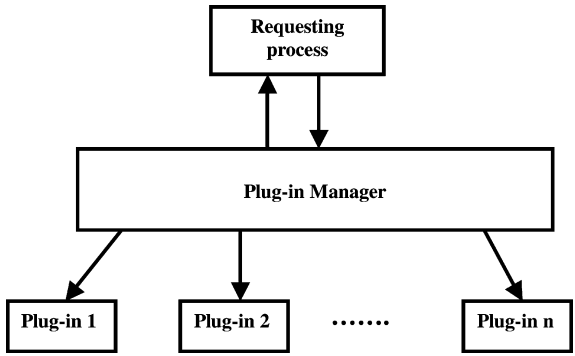


FIG. 13. High-level design of a plug-in architecture.

Upon completion of the service, the plug-in manager component regains control and returns the results to the requesting process. If a system is large and contains different types of functionality, there may be multiple plug-in manager components. In this case, each plug-in manager would be responsible for a subset of the plug-in components.

Requesting Process: The requesting process is the owner of the plug-in manager component(s). The process requests services from a specific plug-in or multiple plug-in components through the plug-in manager component's interface. Upon completion of the service, the results are returned to the requesting process through the plug-in manager component.

In this architectural style, the plug-in manager component and the component that implements the requesting process do not know the actual implementation of each plug-in component. The plug-in manager does not know the details of each plug-in component. It only knows that the plug-in components implement a well-defined interface. This aspect of the plug-in architectural style facilitates the expansion and modification of an application's services. As new functionality is required, a new plug-in component that implements the well-defined interface is created and becomes immediately available to a requesting process through the plug-in manager component. Likewise, components can be exchanged to adapt the behavior of an application. A visual representation of the plug-in architecture is shown in Fig. 13.

5.5.2.2 Design Patterns. The Observer design pattern played a major role in the architectural style of this system. As with the evaluations of the VQI, the concept of an Observer design pattern had to be abstracted to a component level. Instead of having classes play the roles in the design pattern, the roles are played by components that may consist of several classes.

Observer. The Observer is one of the behavioral design patterns described in [19]. It is similar to aspects of the model-view controller paradigm [22] and typically used to provide various views of the same subject data. For example, a data set containing percentages might be viewed in a bar chart, in tabular form, and/or as a pie chart. The subject data in this example is the data set of percentages. The bar chart, tabular form, and pie chart represent various views of the subject data.

In the Observer design pattern, there are Subject classes and Observer classes. Subject classes are the classes that contain the data values that will be observed. Each Observer class interested in a Subject subscribes to be an observer of the Subject. A Subject class contains a list of all of the Observers interested in its data. Whenever the Subject data changes, the Subject class informs all of its Observers that it has changed. The Observer classes adjust themselves according to the change in the Subject data. Using the previous example, the bar chart, table, and pie chart classes would subscribe to be observers of the class containing percentages data. If a user updates the subject data set in the tabular view, a message would be sent to each subscriber in the observer list of the class containing the percentages data set. Upon receipt of the change message, the bar and pie charts would update their views accordingly.

The Observer design pattern is implemented as part of the Java Software Development Kit (SDK). The role of the Subject in the design pattern is implemented in the Observable class. To create Subject classes, developers create subclasses of the Observable class. The role of the Observer class in the design pattern is accomplished through the use of the Observer interface in the Java SDK. Any class that wishes to be an Observer to an Observable object (i.e., the Subject) must implement the Observer interface.

5.5.2.3 The System Context. The diagram shown in Fig. 14 represents the planned architecture of the system under evaluation. The system consists of a Main component, several plug-in components and several library-oriented components. The Main component of the system allows for textual and graphical views of CAD designs and graphical views of simulation results that are used in analysis. The plug-in components fall into two main categories: those related to the CAD design views and those related to the analysis views. The PlugInCommon component contains classes related to the plug-in architectural style. In terms of the plug-in architectural style, the PlugInCommon component is part of the plug-in manager. The Results component contains the data structures representing the data that is manipulated in the analysis operations of the application. Hence, the Results component contains classes that participate in the Observer pattern. The classes in this component represent Subject components in the terms of the Observer pattern. The PlugInMainStructure component contains data structures used in both the textual and

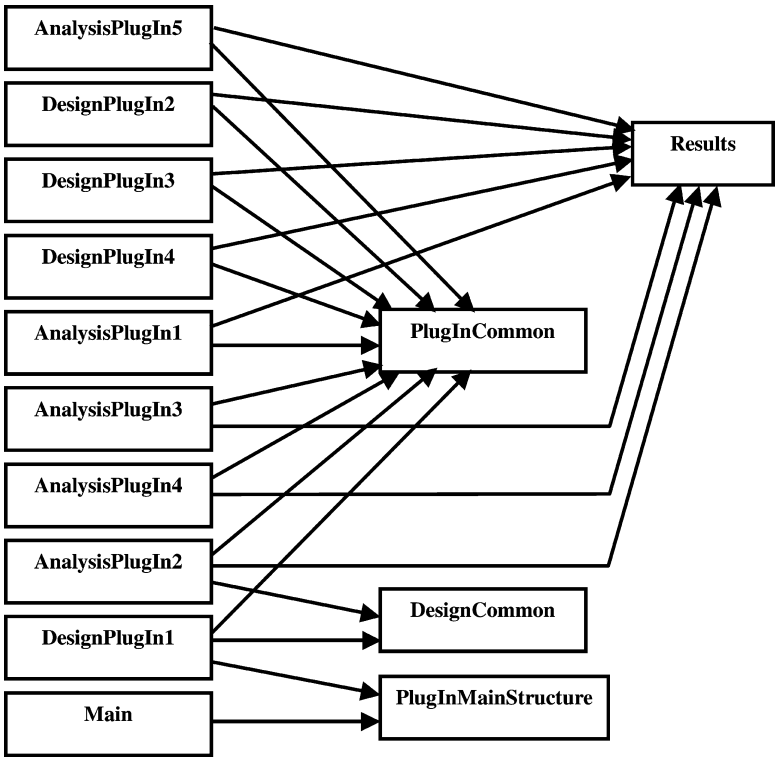


FIG. 14. The planned architecture of the proprietary system.

graphical views of CAD designs. The DesignCommon component contains additional data structures used by several plug-in components. The following components are considered library-oriented components: Results, PlugInMainStructure, and DesignCommon. These components contain data structures that will be accessed by several components. However, these library-oriented components should not access classes in the plug-in components.

The Main component of the planned architecture houses the requesting process for the plug-in architectural style. The system consists of a single plug-in manager component. This plug-in manager component is partially implemented in the PlugInCommon component and partially implemented in the Java SDK. Specifically, as long as a plug-in component is a Java GUI component or a subclass of a Java GUI component and implements the Observer interface, it will implement the well-defined interface required by the plug-in manager component for this system.

Because part of the architectural style and the design patterns used in this system are contained within the implementation of the Java SDK, the diagrams of the planned and actual architectures will not show the complete architectural style and design pattern. For example, the connection between the Main component and the plug-in manager component will not appear in the diagrams. These peculiarities have to be taken into consideration when determining the architectural and metric guidelines for the system.

5.5.3 Guidelines

Several architectural and metric guidelines were developed based on the architectural style and design patterns used in the system.

5.5.3.1 Architectural Guidelines. The architectural guidelines for the system are broken into guidelines related to each type of component in the architectural style.

Plug-in Component Guidelines:

- A plug-in component cannot access other plug-in components directly. All communication between plug-in components must go through the plug-in manager component.
- No component (including other plug-in components) should reference plug-in components. The part of the plug-in manager component that would access the plug-in components is part of the Java SDK and is not visible in the diagram of the planned architecture.

Plug-in Manager Component Guidelines:

- In most cases, plug-in components may access the PlugInCommon and the Results components only. The following components are exceptions to this rule: DesignPlugIn1 and AnalysisPlugIn2. These plug-in components are allowed to access the DesignCommon component. DesignPlugIn1 is allowed to access classes in the PlugInMainStructure component as well.
- Non-plug-in components should not reference the PlugInCommon component.
- The PlugInCommon component should not reference non-common components.
- Plug-ins must extend from a class in the PlugInCommon component. In the diagram of the planned architecture for the system, the PlugInCommon represents part of the plug-in manager module. It contains the classes and interfaces needed by plug-in components.

Component Housing Requesting Process Guidelines:

- The Main component cannot access plug-ins directly. Communication must go through the plug-in manager component. Since the communication between the Main component and the plug-in manager module is implemented in Java SDK classes, there should not be any references from the Main component to the plug-in or plug-in manager components in the diagrams.

Library-Oriented Component Guidelines:

- Library-oriented components are allowed to access each other, but they are not allowed to access plug-in components or non-library components. Although coupling among library-oriented components is allowable, this type of coupling is sometimes a violation. For example, in the planned architecture, there is no arrow between the PlugInMainStructure component and the PlugInCommon component. If classes in the PlugInMainStructure component access classes in the PlugInCommon component, it would be a violation.
- Library-oriented components are designed to be used by many components; hence coupling to the library-based components is expected to be high.

5.5.3.2 Metric Guidelines. In addition to the architectural guidelines, the following metric guidelines were defined for the system:

- The CBM metric of component PlugInCommon should be nine (one for each of the nine plug-in components). The CBMC metric of PlugInCommon component should be high in relative to other components.
- The CBM metrics for most plug-in components should be two. The two exceptions are the DesignPlugIn1 and AnalysisPlugIn2 components.
- Communication between the plug-in manager module (PlugInCommon) and the plug-in components should be through a common, narrow interface. The CBMC metric for plug-in components should be low and the CBMC metric for the PlugInCommon should be relatively high since it coordinates all plug-in components.
- The CBM metric for the Main component should be one due to the fact that the connection to the plug-in manager module is part of the Java SDK and not visible in the diagrams. The Main component should have a dependency with the PluginMainStructure component only.
- The CBM metric of the Results component should be at least nine (one for each plug-in component).
- The CBMC metric of the library-oriented components should be high relative to other components.

5.5.4 Violations Between the Planned and Actual Architectures

Once the guidelines were established, the comparison between the planned and actual architectures was conducted. To extract the static couplings and to make the comparison, a new tool was developed that built on the functionality provided by the existing tool. This tool is called the architectural evaluation tool. Initially, the tool uncovered several false positives. In particular, the source code for this system includes several classes with the same names as classes defined within the Java 3D graphics package. The tool mistakenly detected couplings between components that were using the Java 3D graphics classes and the component where the identically named classes were defined. This type of false positive suggests an area of improvement for the architectural evaluation tool.

The false positive violations were removed and Fig. 15 shows the comparison between the planned and actual architectures of the system with the remaining violations highlighted. The missing coupling type violations are depicted with dotted lines and the extra coupling type violations are shown with dashed lines. In total, once the false positive violations were removed, there were 24 violations in the comparison between the planned and actual architectures. Of the 24 violations, 21 were extra coupling violations while three were missing coupling violations.

5.5.4.1 Discussion of Types of Violations. Once the false positive violations were removed, a number of violations remained. The remaining violations varied in terms of severity. Some were minor violations while others suggested more serious problems with the system's actual architecture.

Minor Violations. As with the VQI evaluations, one of the violations uncovered in this study was due to an unused import statement. Initially, a class was imported and used within the component. However, the code was changed to no longer rely on the external class and the import statement was inadvertently left in the code. Another violation was due to dead code. Apparently, a class that was used in earlier versions of the system remained packaged with the source code. This class was coupled to a class in an external component. It was not until after the evaluation was conducted that the developers realized that the class was still included in the source code.

Medium-Level Violations. Several of the violations uncovered during the evaluation were due to coupling in test methods that were not actually part of the system. As a way of testing individual classes, often a test main method was included as part of the class. In several cases, these test methods used classes from external components, creating coupling between components. Although these test methods are not

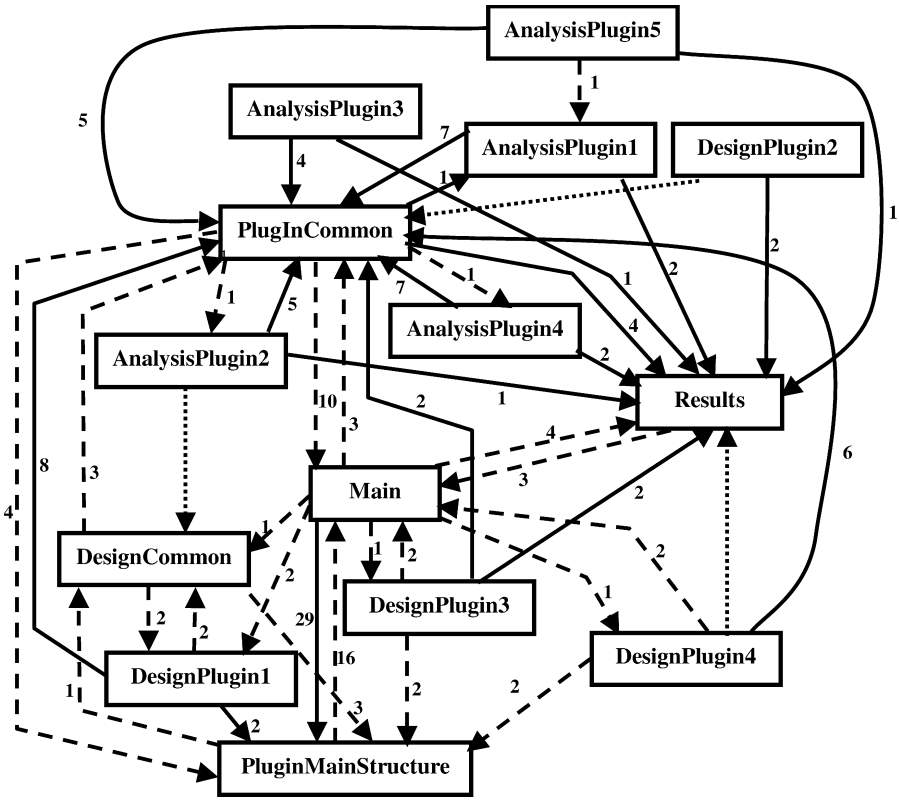


FIG. 15. The comparison between the planned and actual architectures. Missing coupling violations are depicted with a dotted line. Extra coupling violations are depicted with a dashed line. The label on each arc represents the number of classes accessed in the component at the endpoint of the arrow.

part of the actual system, having the dependencies to external components does pose a potential maintenance problem in the future. Hence, these violations have been listed as medium-level violations.

Major Violations. The architectural evaluation uncovered several major violations. A violation was considered to be major if it violated one of the architectural style or design pattern guidelines due to a misunderstanding of the style or pattern. As a result of the evaluation, it was discovered that several of the design plug-ins were not being connected correctly in the application. All plug-in components are supposed to extend from a class in the PlugInCommon component and utilize data

structures within the Results component. However, some of the design plug-in components were using the wrong data structures (i.e., those from the PlugInMainStructure component instead of the Results component) and accessing classes in the Main component directly. Additionally, one of the design plug-in components (DesignPlugIn2) did not utilize any classes in the PlugInCommon component as expected. These extra and missing dependencies also indicate a misunderstanding of how plug-in components are supposed to be connected in the application.

There were fewer, yet still some problems with analysis plug-in components as well. The extra dependency between the PlugInCommon component and the AnalysisPlugIn4 component represents another potentially serious violation. The PlugInCommon component is not supposed to access classes in the plug-in components. However, the PlugInCommon has a dependency with the AnalysisPlugIn4 component that may cause problems as the development of the product continues.

5.5.4.2 Metric Analysis. An evaluation of the architecture using the CBM and CBMC metrics provides additional information about how well the actual architecture meets the guidelines of the planned architecture. The CBM and CBMC metrics for the system under evaluation indicate several problems. The CBM and CBMC values for the components of the system are shown in Table V. The charts in Fig. 16 show the distribution of CBM and CBMC values.

TABLE V
THE CBM AND CBMC VALUES
OF THE COMPONENTS OF THE SYSTEM

Module	CBM	CBMC
DesignPlugIn2	1	2
AnalysisPlugIn3	2	5
AnalysisPlugIn2	2	7
AnalysisPlugIn5	3	7
AnalysisPlugIn4	2	10
DesignPlugIn4	3	10
AnalysisPlugIn1	3	11
DesignCommon	4	11
DesignPlugIn3	4	14
DesignPlugIn1	4	15
Results	9	18
PlugInMainStructure	6	48
PlugInCommon	12	55
Main	7	66

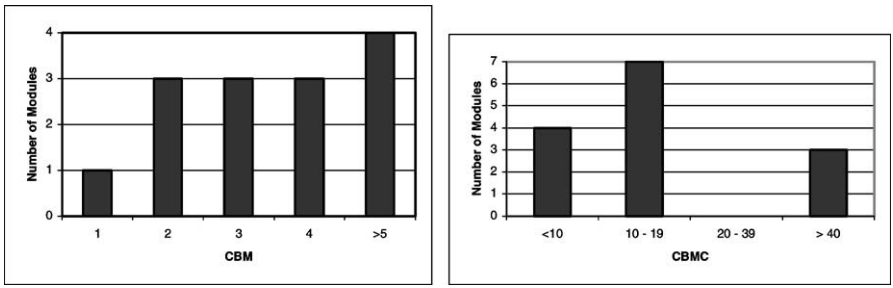


FIG. 16. The distribution of the CBM values is shown in the chart on the left. The distribution of the CBMC values is shown in the chart on the right.

The CBM metrics for the plug-in components confirm the discrepancies seen in the comparison diagram. (See Fig. 16.) Six out of the nine plug-in components violate the metric guidelines set for the expected CBM values. Five of the six substandard components have a higher CBM value than expected while one has a lower CBM value than expected. Additionally, the CBM values for the PlugInCommon and Main components are 12 and 7 respectively, much higher than expected. The high CBM values indicate that the plug-in components have not been connected to the application correctly and that there are unwanted dependencies among the plug-in components as well as with the Main component. In the future, if these violations are not corrected, there may be maintainability problems due to the higher than expected coupling between the components.

Looking at the CBMC numbers provides a more detailed view of the architecture than can be seen in the diagrams or with the CBM numbers alone. Since the CBMC number reflects the number of classes involved in dependencies with the component, it gives a better understanding of the degree of coupling between components. One of the metric rules states that the plug-in components should have relatively low CBMC values. Several plug-in components have CBMC values as high as the library-oriented components where high CBMC values are expected. In many cases, the contributors to the high values come from dependencies with classes in library-oriented components. However, after examining the classes involved in the dependencies with plug-in components further, it can be seen that the interface between the PlugInCommon component and the plug-in components is not as narrow as expected. Perhaps, the metric guideline needs to be adjusted to take into account the extra references. Alternatively, narrowing the interface between plug-in components and the PlugInCommon component should be considered.

The CBM and CBMC numbers for the Main component raise concerns about the maintainability of this component. At 66, the Main component has the highest CBMC value and its CBM value of 7 is much higher than the expected value of 1.

This component clearly violates the planned architecture and should be corrected to avoid more serious maintenance problems in the future.

5.5.5 Summary of Results

The study was useful on several different levels. The development and analysis team gained insights during the evaluation. Several issues with the evaluation process were addressed and validated with this study.

The evaluation provided useful feedback on the status of the software architecture for the development team. The system used in this evaluation was still under development when the study was conducted. The architectural evaluation process highlighted several problems that were unknown to the developers of the system prior to the study. For example, after the evaluation, it was clear that there was a misunderstanding between the developers regarding the connection of design plug-in components to the application. The developers of the system found the architectural evaluation process to be practical and helpful.

The evaluation of the system provided validation for several different aspects of the software architectural evaluation process. This study represented the first time the process had been used on software that was not originally known to the evaluators. In the evaluations of the VQI, some of the evaluators were also responsible for the design and development of the system. This study validated that it is feasible for the architectural evaluation process to be used with a system where the evaluators are not involved with the design or development of the system under study.

The proprietary commercial system was larger than the VQI systems evaluated previously. There were no difficulties in applying the process and the architectural evaluation tool to the larger system. This study demonstrated that the architectural evaluation process could be used on Java software systems with size up to approximately 100 KLOC. There is no indication that the process and the tool will not work on larger systems; however, this proprietary system is the largest one to date where the process and tool have been used.

The system under evaluation used a different architectural style and additional design patterns from the VQI. The study confirmed that the architectural evaluation process can be adapted to new architectural styles and design patterns relatively easily.

In addition to providing validation for several new aspects of the software architectural evaluation process, this study also provided feedback on the process and the tools. The system developers were happy with the results of the evaluation. In the process of evaluating the system, several new features related to the tool support were discussed. For example, during the evaluation of this system it became clear how to remove some of the false positive violations.

6. Summary and Future of Architectural Evaluation

As software systems have become more complex, the need to study and evaluate these systems at a higher level of abstraction has become more important. Viewing the high-level architecture of a system allows for an evaluation that focuses on the components and the interactions between components. Architectural styles are common, reusable high-level architectural designs. Similarly, at a lower level of abstraction, design patterns represent another form of reusable design. By choosing architectural styles and design patterns, there are several implicit and explicit rules and guidelines for a system. It is important to understand the implications of selecting a particular architectural style and/or design patterns. In many cases, these implications are not explicitly stated or documented in the descriptions making these styles and patterns difficult to implement correctly. To avoid or detect architectural mismatches, the implications have to be made explicit.

Software architectural evaluation techniques are used to reason about the choices made for the architecture of a system. There are two main categories of software architectural evaluations, those that are done before a system has been implemented and those that are conducted after a version of the system exists. Both forms of evaluation are useful for similar, but different, reasons. Pre-implementation software architectural evaluations are useful for choosing an architectural style and evaluating the adequacy of the architectural choices. Architectural evaluations performed after a version of the system has been implemented are used to track the actual implementation of the system. They are useful in determining whether or not the system has been implemented as planned.

The software architectural evaluation process described in this chapter is an implementation-oriented approach designed to be efficient. This approach has been applied successfully to several systems varying in size, architectural styles and design patterns. The approach has been useful in highlighting architectural mismatches and in identifying potential problem areas of the software system.

Several tools have been developed specifically to support the process described in this chapter. However, other existing tools might be used instead of or in combination with the architectural evaluation tools used to conduct the case studies. For example, it would be interesting to combine the approach from Antoniol et al. [2] with the process and tools described in this chapter. The approach described in [2] attempts to identify design patterns based on metrics of the code. Perhaps using this approach and tool might improve the process of retrieving the actual architecture from the code.

Identifying components from the source code and the expected interrelationships among them is another area for continued effort. With the existing tool set, this part of the process is done manually. Automating the mapping from source code organi-

zation to component organization is not a trivial problem. In languages such as Java, it might be easier to identify components if the code is developed with packages. However, not all languages have the concept of a package and identifying the pieces of source code that represents a component becomes more difficult.

The results from using this software architectural evaluation approach have been encouraging. In every case where the approach has been used, valuable feedback has been given to the development team. However, in all cases, maintainability was the perspective chosen for evaluation. It would be interesting to identify metrics and guidelines from a different perspective. One possible perspective could be security. Inconsistencies between the planned and actual implementation of software architectures is a source of many security vulnerabilities [33]. Establishing metrics, guidelines and rules from a security perspective with the architectural evaluation approach might be useful in finding these architectural mismatches. As more experience in software architectural evaluations is gained, more knowledge about software architecture can be gathered.

ACKNOWLEDGEMENTS

We thank David Cohen, Tristan Everitt, and Liliya Kharevych for implementing tools to support the architectural evaluation process. Lorin Hochstein was very helpful in researching various software architecture technologies. We also thank John Tvedt for providing access to the commercial system used in the case study and for his feedback on the process. We also thank Jen Dix for proofreading the document.

REFERENCES

[1] Abowd G., Bass L., Clements P., Kazman R., Northrop L., Zaremski A., *Recommended Best Industrial Practice for Software Architecture Evaluation*, Software Engineering Institute, 1996, CMU/SEI-96-TR-025.

[2] Antoniol G., Fiutem R., Cristoforetti L., "Using metrics to identify design patterns in object-oriented software", in: *Proceedings of the 5th International Symposium on Software Metrics*, 1998, pp. 23–34.

[3] Avritzer A., Weyuker E.J., "Investigating metrics for architectural assessment", in: *Proceedings of the 5th International Symposium on Software Metrics*, 1998, pp. 4–10.

[4] Basili V.R., Caldiera G., Rombach D.H., "The experience factory", in: *Encyclopedia of Software Engineering. 2 Volume Set*, 1994, pp. 469–476.

[5] Basili V.R., Caldiera G., Rombach D.H., "The goal question metric approach", in: *Encyclopedia of Software Engineering. 2 Volume Set*, Wiley, 1994, pp. 528–532.

[6] Bass L., Clements P., Kazman R., *Software Architecture in Practice*, Addison–Wesley, 1997.

[7] Beck K., *Extreme Programming Explained: Embrace Change*, Addison–Wesley, 1999.
[8] Bengtsson P., Bosch J., "Architecture level prediction of software maintenance", in: *Proceedings of 3rd EuroMicro Conference on Maintenance and Reengineering*, 1999, pp. 139–147.
[9] Bhansali S., "Software design by reusing architectures", in: *Proceedings of the 7th Knowledge-Based Software Engineering Conference, McLean, Virginia*, September 1992.
[10] Booch G., Rumbaugh J., Jacobson I., *The UML Modeling Language User Guide*, Addison–Wesley, 1999.
[11] de Bruijn H., van Vliet H., "Scenario-based generation and evaluation of software architectures", in: *Proceedings of the 3rd International Conference on Genative and Component-Based Software Engineering*, September 2001, pp. 128–139.
[12] Chidamber S.R., Kemerer C.F., "Towards a metrics suite for object-oriented design", *Transactions on Software Engineering* **20** (6) (1994) 476–493.
[13] Clements P., Kazman R., Klein M., *Evaluating Software Architectures: Methods and Case Studies*, Addison–Wesley, 2001.
[14] Clements P., Kogut P., "The software architecture renaissance", *Crosstalk, The Journal of Defense Software Engineering* **7** (1994) 20–24.
[15] Crispen R., Stuckey L., "Structural model: Architecture for software designers", in: *Proceedings of TRI-Ada '94, Baltimore Convention Center, Baltimore, MD*, November 1994.
[16] Deremer F., Kron H., "Programming in the large versus programming in the small", *IEEE Transactions on Software Engineering* **2** (2) (June 1976) 80–87.
[17] Dobrica L., Niemela E., "A survey on software architecture analysis methods", *IEEE Transactions on Software Engineering* **28** (7) (2002).
[18] Eick S.G., Graves T.L., Karr A.F., Marron J.S., Mockus A., "Does code decay? Assessing the evidence from change management data", *IEEE Transactions on Software Engineering* **27** (1) (2001) 1–12.
[19] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns Elements of Reusable Object-Oriented Software*, Addison–Wesley, 1994.
[20] Garlan D., Shaw M., "An introduction to software architecture", in: *Proceedings of Advances in Software Engineering and Knowledge Engineering*, 1993, pp. 1–39. (Also Technical report, CMU-CS-94-166, January 1994).
[21] Hochstein L., Lindvall M., "Matching architectural needs to technologies: A survey", Technical report, FC-MD 03-113, January 2003.
[22] Krasner G.E., Pope S.T., "A description of the model-view-controller user interface paradigm in the Smalltalk-80 system", *Journal of Object Oriented Programming* **1** (3) (1988) 26–49.
[23] Kruchten P.B., "The 4 + 1 view model of architecture", *IEEE Software* **12** (6) (November 1995).
[24] Lassing N., Bengtsson P., van Vliet H., Bosch J., "Experiences with ALMA: architecture-level modifiability architecture analysis", *Journal of Systems and Software* (2002) 47–57.

[25] Lindvall M., Tesoriero R., Costa P., "Avoiding architectural degeneration: An evaluation process for software architecture", in: *Proceedings of the International Symposium on Software Metrics*, 2002, pp. 77–86.

[26] Lindvall M., Tesoriero Tvedt R., Costa P., "An empirically-based process for software architecture evaluation", *Journal of Empirical Software Engineering* **8** (1) (March 2003) 83–108.

[27] Murphy G., Notkin D., Sullivan K., "Software reflexion models: Bridging the gap between source and high-level models", in: *Proceedings of SIGSOFT'95 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 1995, pp. 18–28.

[28] Perry D., Wolf A., "Foundations for the study of software architecture", *ACM Sigsoft—Software Engineering Notes* **17** (4) (October 1992).

[29] Schwanke R.W., "An intelligent tool for reengineering software modularity", in: *Proceedings of the 13th International Conference on Software Engineering*, 1991, pp. 83–92.

[30] Seaman C., de Mendonca N.M.G., Basili V.R., Kim Y.-M., "An experience management system for a software consulting organization", in: *24th NASA SEL Software Engineering Workshop (SEW'24)*, 1999.

[31] Soni D., Nord R., Hofmeister C., "Software architecture in industrial applications", in: *Proceedings of the International Conference on Software Engineering, Seattle*, 1995.

[32] Tesoriero Tvedt R., Costa P., Lindvall M., "Does the code match the design? A process for software architectural evaluation", in: *Proceedings of the International Conference on Software Maintenance*, 2002.

[33] Viega J., McGraw G., *Building Secure Software: How to Avoid Security Problems the Right Way*, Addison–Wesley, 2001.

[34] Yacoub S.M., Ammar H., "A methodology for architectural-level reliability risk analysis", *IEEE Transactions on Software Engineering* **28** (6) (2002) 529–547.

This Page Intentionally Left Blank

Efficient Architectural Design of High Performance Microprocessors

LIEVEN EECKHOUT AND KOEN DE BOSSCHERE

Department of Electronics and Information Systems (ELIS)
Ghent University
Sint-Pietersnieuwstraat 41
B-9000 Gent
Belgium
leeckhou@elis.UGent.be
kdb@elis.UGent.be

Abstract

Designing a high performance microprocessor is extremely time-consuming taking at least several years. An important part of this design effort is architectural simulation which defines the microarchitecture or the organization of the microprocessor. The reason why these simulations are so time-consuming is fourfold: (i) the architectural design space is huge, (ii) the number of benchmarks the microarchitecture needs to be evaluated with, is large, (iii) the number of instructions that need to be simulated per benchmark is huge as well, and (iv) simulators are becoming relatively slower due to the increasingly complex designs of current high performance microprocessors. In this chapter, we extensively discuss these issues and for each of them, we propose a solution. As such, we present a new simulation methodology for designing high performance microprocessors. This is done by combining several recently proposed techniques, such as statistical simulation, representative workload design, trace sampling and reduced input sets. This chapter presents a holistic view on speeding up the architectural design phase in which the above mentioned techniques are integrated in one single architectural design framework. In this methodology, we first identify a region of interest in the huge design space through statistical simulation. Subsequently, this region is further explored using detailed simulations. Fortunately, these slow simulations can be sped up: (i) by selecting a limited but representative workload, (ii) by applying trace sampling and reduced input sets to limit the simulation time per benchmark, and (iii) by optimizing the architectural simulators. As such, we can conclude that this methodology can reduce the total simulation time considerably. In addition to presenting this new architectural modeling and simulation

ADVANCES IN COMPUTERS, VOL. 61
ISSN: 0065-2458/DOI 10.1016/S0065-2458(03)61002-8

Copyright © 2004 Elsevier Inc.
All rights reserved.

approach, we present a survey of related work of this important and fast growing research field.

1. Introduction

An important phenomenon that can be observed nowadays is the ever increasing complexity of computer applications. This trend is primarily caused by the high performance of modern computer systems (Moore's law) and by the high demands of end users. The increasing performance of computer systems is made possible by a number of factors. First, today's chip technologies can integrate several hundreds of millions of transistors on a single die. In addition, these transistors can be clocked at ever increasing frequencies. Second, computer architects develop advanced microarchitectural techniques to take advantage of these huge numbers of transistors. As such, they push the performance of microprocessors even further. Third, current

compilers as well as dynamically optimizing environments are capable of generating highly optimized code.

These observations definitely have their repercussions on the design methodologies of current and near future microprocessors. Nowadays it is impossible to design a high performance microprocessor based on intuition, experience and rules of thumb. Detailed performance evaluations through simulations are required to characterize the performance of a given microarchitecture for a large number of applications. As a result, the total design time of a complex microprocessor can take up to seven long years, see for example Mukherjee et al. [59]. During this design process we can identify several design steps as discussed by Bose and Conte [7], Bose et al. [8] and Reilly [69]:

(1) the selection of a workload or choosing a number of representative benchmarks with suitable inputs.
(2) design space exploration or bounding the space of potential designs by using rough estimates of performance, power consumption, total chip area, chip packaging cost, pin count, etc.
(3) architectural simulations which define the microarchitecture or the internal organization of the microprocessor, for example, the number of arithmetic-logical units (ALUs), the size of the caches, the degree of parallelism in the processor, etc.
(4) register transfer level (RTL) simulations modeling the microprocessor at a lower abstraction level which incorporates full function as well as latch-accurate pipeline flow timing.
(5) logic and gate level simulations in which the functioning of the microarchitecture is modeled at the level of NAND gates and latches.
(6) circuit level simulations which model the microprocessor at the chip layout level.
(7) and finally, verification, see for example [4].

In this chapter, we will limit ourselves to the first three design steps. These three steps actually define the optimal microarchitecture for a given set of benchmarks. Note that although these design steps are done at a high level, they are still extremely time-consuming, especially step 3, the architectural simulations.

There are four reasons why architectural simulations are so time-consuming. First, the microprocessor design space is huge but needs to be explored in order to find the optimal design. Note that the design space is limited due to a number of design constraints such as maximum chip area, maximum power consumption, maximum power density, maximum temperature, etc. However, even under these design constraints, the microprocessor design space remains huge. Second, the workload space, i.e., the number of computer programs to be simulated, is large as well. For example,

the SPEC CPU2000 suite collects 26 CPU-intensive benchmarks. Note that for several of these benchmarks several inputs are provided making this number even larger. Third, the number of instructions that need to be simulated per computer program is enormous. Evaluating the performance of complex applications requires the simulation of huge numbers of instructions—nowadays, tens or even hundreds of billions of instructions are simulated per application. Fourth, the simulators themselves are slow since they need to model complex microarchitectures. For example, SimpleScalar's widely used detailed architectural simulator requires on average 300,000 simulator instructions per simulated instruction, see [1].

In this chapter, we present a new simulation methodology that considerably reduces the total architectural simulation time by addressing these four issues. This is done by first exploring the entire design space and selecting a region with interesting properties in terms of performance and/or power consumption. Statistical simulation, which is a fast and quite accurate simulation technique, is well suited for performing this design step. Subsequently, simulations need to be run on detailed and thus slow simulators to identify the optimal design within this region of interest. These slow microarchitecture-level simulations can be sped up (i) by selecting a limited set of representative benchmarks, (ii) by applying trace sampling and by using reduced input sets to limit the number of instructions per benchmark and (iii) by optimizing the architectural simulators to increase their instruction throughput. As such, using this simulation methodology the architectural design phase can be sped up considerably without losing accuracy, i.e., the same optimal design is identified as would have been the case through common practice long-running architectural simulations.

This chapter is organized as follows. Section 2 gives a brief introduction to out-of-order microarchitectures which are commonly used in contemporary general-purpose microprocessors. Section 3 details on current practice in architectural design. In Section 4, we discuss why architectural simulations are so extremely time-consuming. In Section 5, we present our new simulation methodology which reduces the total architectural simulation time by addressing the following four issues: the huge design space, the large workload space, the number of instructions per benchmark and the simulator slowdown factor. Each of these issues are discusses in the following sections: 6 through 9. With each of these, we extensively discuss related work. Finally, we conclude in Section 10.

2. Out-of-Order Architecture

Most contemporary general-purpose microprocessors implement an out-of-order microarchitectural organization. Examples of commercial out-of-order microprocessors are the Alpha 21 264, see [46], the Pentium 4, see [39], and the MIPS R10000,

see [87]. Many of the basic mechanisms of an out-of-order architecture were proposed by Tomasulo [83] for the design of the IBM 360 model 91 in 1967. This section gives a brief introduction to out-of-order architectures. For a more elaborate description we refer the interested reader to [77] and [37]. In an out-of-order architecture, see Fig. 1, instructions are fetched from the instruction cache (I-cache). In case branch instructions are fetched, the branch target and/or the branch direction are predicted so that in the next cycle instructions can be fetched along the predicted path. Fetched instructions are subsequently renamed by the register renaming unit. Register renaming eliminates write-after-read (WAR) and write-after-write (WAW) dependencies from the instruction stream; only real read-after-write (RAW) data dependencies remain. Once the instructions are transformed into a static single assignment form, they are dispatched to the *instruction window*, where the instructions wait for their source operands to become available (data-flow execution). Each clock cycle, ready instructions are selected from the instruction window to be executed on a functional unit. The number of instructions that can be selected for execution in one clock cycle, is restricted to the *issue width*. Further, bypassing is implemented which means that data-dependent instructions can be executed in consecutive cycles. Once an instruction is executed, the instruction can be *retired* or removed from the processor core. At retirement time, the results of the instructions are written to the register file or memory. To preserve the semantics of a program trace, the instructions are retired sequentially as they were fetched. The number of instructions that can be retired in one clock cycle, is restricted to the *reorder width*.

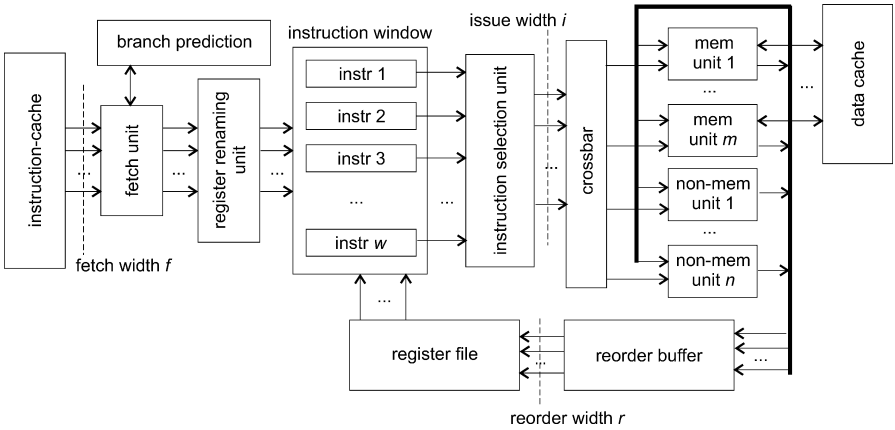


FIG. 1. Out-of-order architecture.

3. Current Practice

Before going into detail on why architectural simulation is so time-consuming, we first give an overview of current practice. We identify four simulation approaches: functional simulation, specialized cache and branch predictor simulation, trace-driven simulation and execution-driven simulation.

Functional simulation is the simplest form of simulation and models the functional behavior of an instruction set architecture (ISA). This means that instructions are simulated one at a time by taking their input operands and producing output values. In other words, a functional simulator basically is an ISA emulator. Functional simulation is extremely useful for determining whether a software program or operating system is implemented correctly, see for example Simics, a product of Virtutech, as described by Magnusson et al. [52]. Functional simulators are also extremely useful for architectural simulation since these tools can generate instruction and address *traces* which can be used by other tools in the design flow. A trace is a linear sequence of instructions that a computer program produces when it gets executed. The length of such a trace is called the *dynamic instruction count* of the application.

Specialized cache and branch predictor simulators take instruction and address traces as input and simulate cache behavior and branch prediction behavior in isolation. The performance metric that these tools typically produce is *miss rates*, or the number of cache misses of branch mispredictions per access to the cache and the branch predictor, respectively. These tools are widely available, for example cheetah by Sugumar and Abraham [82] and DineroIV by Edler and Hill [20].

Trace-driven simulation also takes as input instruction and address traces but simulates a complete microarchitecture in detail instead of isolated units. Since the microarchitecture is modeled in a more or less cycle-accurate way, this kind of simulation is also referred to as *timing simulation*. As such, a trace-driven simulation methodology separates functional simulation from timing simulation. This approach has the important benefit that functional simulation needs to be done only once whereas timing simulation has to be done multiple times to evaluate various processor configurations. This can reduce the total simulation time. An important disadvantage is that these traces need to be stored on a disk. These traces can be very large since the number of instructions that need to be stored in a trace file equals the dynamic instruction count. For current applications, the dynamic instruction count can be several hundreds of billions of instructions. Storing these huge trace files might be impractical in some situations. Another disadvantage is of particular interest when modeling current out-of-order microprocessors. Out-of-order microarchitectures predict the outcome of branches and speculatively execute instructions along the predicted path. In case of a correct prediction, this will speed up the execution of the application. In case of a misprediction, these speculatively executed instructions

need to be nullified. These speculatively executed instructions however, do not show up in a trace file and as such, do not get simulated in a trace-driven simulator. These instructions however can have a significant impact on overall performance because they require resources that need to be shared with instructions along the correct path. Also, these speculatively executed instructions can result in prefetching effects or cache contention, see [2].

Execution-driven simulation is similar to trace-driven simulation but combines functional simulation with timing simulation. As a consequence, trace files do not need to be stored and speculatively executed instructions get simulated accurately. In recent years, execution-driven simulation has become the method of choice. A well known example of an execution-driven simulator is SimpleScalar's out-of-order simulator described by Burger and Austin [11], and Austin et al. [1]. This simulator is widely used in the academia in the field of computer architecture research. Other simulation tools that were designed in the academia are Rsim at Rice University by Hughes et al. [41], SimOS at Stanford University by Rosenblum et al. [71], fMW at Carnegie Mellon University by Bechem et al. [2] and TFsim at the University of Wisconsin–Madison by Mauer et al. [55]. Companies have similar tools, for example Asim described by Reilly and Edmondson [70] and Emer et al. [27] and used by the Compaq design team which is now with Intel, and MET used by IBM and described by Moudgill [57] and Moudgill et al. [58]. Note that due to the fact that these simulators do model a microarchitecture at a high abstraction level, discrepancies might occur when their performance results are compared to real hardware, see for example Black and Shen [3], Gibson et al. [31], Desikan et al. [17].

4. Architectural Simulation

This section discusses why architectural simulations are so time-consuming. Note that this is irrespective of whether trace-driven simulation or execution-driven simulation is used.

4.1 Design Space

Designing a microprocessor with optimal characteristics can be viewed as evaluating all possible configurations and selecting the optimal one. As such, the total simulation time T is proportional to the number of processor configurations P that need to be evaluated: $T \sim P$. It is interesting to note that the optimal microprocessor configuration depends on the design criteria. For example, in a workstation, performance will obviously be the major concern. On a mobile device on the other hand, energy consumption will be the key design issue.

Irrespective of the primary design concern, the design space is obviously huge since there are typically several dozens or even over one hundred architectural parameters that need to be tuned. Some examples of architectural parameters are: the number of ALUs, the types of the ALUs, the latencies of the ALUs, the number of instructions executed per cycle, the number of memory ports, the configuration of the memory hierarchy, etc. And since every architectural parameter can take several values—e.g., the number of ALUs can be varied from 2 up to 8—the total number of processor configurations that need to be evaluated is exponential in the number of architectural parameters. For example, if we assume there are 100 architectural parameters each possibly taking 4 values, the total number of processor configurations that need to be evaluated is $4^{100} \approx 1.6 \cdot 10^{60}$. As mentioned in the introduction, some of these designs cannot be realized due to design constraints, such as maximum chip area, maximum power consumption, etc. However, even under these constraints, the search space will still be huge. As such, it is obvious that we need better methods than enumeration to reduce the total number of processor configurations that need to be evaluated.

4.2 Workload Space

Evaluating the performance of a microprocessor configuration is done by simulating a collection of computer programs or benchmarks with suitable inputs. This collection of benchmarks is called the *workload*.

The composition of a workload requires that benchmarks with suitable inputs are selected that are representative for the target domain of operation of the microprocessor, see [37]. For example, a representative workload for a microprocessor that is targeted for the desktop market will typically consist of a number of desktop applications such as a word processor, a spreadsheet, etc. For a workstation aimed at scientific research on the other hand, a representative workload should consist of a number of applications that are computation intensive, e.g., weather prediction, solving partial differential equations, etc. Embedded microprocessors should be designed with a workload consisting of digital signal processing (DSP) and multimedia applications. Note that composing a workload consists of two issues: (i) which benchmarks need to be chosen and (ii) which input data sets need to be selected. It is important to realize that the composition of a workload is extremely crucial since the complete design process will be based on this workload. If the workload is badly composed, the microprocessor will be optimized for a workload that is not representative for the real workload. As such, the microprocessor might not attain the optimal performance in its target domain of operation.

If W is the size of the workload, i.e., the total number of program-input pairs in the workload, we can state that the total simulation time T is proportional to W, or in

other words $T \sim W$. Since we have to evaluate the performance for each processor configuration P, see previous section, for each program-input pair in the workload, the total simulation time becomes proportional to the product of P and W: $T \sim P \cdot W$.

4.3 Length of Program Runs

A simulator of a microprocessor needs to simulate every individual instruction of a program run. As a result, the total simulation time is proportional to the average number of instructions in a single program run I. Consequently, the total simulation time is proportional to $T \sim P \cdot W \cdot I$.

The number of instructions in a single program run is constantly increasing over the years. The reason for this phenomenon is the increasing complexity of todays applications. Indeed, huge numbers of instructions need to be simulated in order to have a workload that is representative for real applications. For example, the Standard Performance Evaluation Corporation (SPEC) released the CPU2000 benchmark suite which replaces the CPU95 benchmark suite, see [38]. The dynamic instruction count of CPU2000 is much higher than CPU95 which is beneficial for real hardware evaluations but infeasible for architectural simulations. For example, the dynamic instruction count of the CPU2000 integer benchmark parser with reference input is about 500 billion instructions. On the other hand, none of the CPU95 integer benchmarks has a dynamic instruction count that is larger than 100 billion instructions.

Another way of looking at the same problem, is as follows. Consider for example one second of execution time on a 2 GHz machine. If we assume that the number of instructions executed per clock cycle (IPC) varies between 1 or 2 on current microprocessors, then we can conclude that one second of a real machine corresponds to 2 to 4 billion instructions. In other words, simulating a representative time window in the order of minutes or hours, requires the simulation of hundreds or even thousands of billions of instructions. Obviously, simulating these huge numbers of instructions for a single program-input pair run is impractical, if not impossible.

4.4 Simulation Cost

Architectural simulators, although they model a microarchitecture at a high level, they are quite slow. Bose [5] reports a simulation cost for the IBM simulation tools of 50,000 instructions per cycle. Austin et al. [1] report a simulation cost of 300,000 instructions per cycle for SimpleScalar's most detailed architectural simulator. If we assume an instruction throughput per cycle (IPC) of 1, we observe that simulation is a factor 50,000 to 300,000 times slower than real hardware simulation. This means

that simulating the above mentioned SPEC CPU2000 benchmark parser with a dynamic instruction count of 500 billion instructions takes three weeks to nearly four months for the above mentioned simulation slowdown factors. There is one prominent reason for this phenomenon, namely the ever increasing complexity of current microprocessor designs. Indeed, computer architects are designing more and more complex microarchitectures in order to get the highest possible performance in a given chip technology. A number of important microarchitectural features have been added to increase performance: branch prediction, speculative execution, memory disambiguation, prefetching, cache line prediction, trace caches, etc. All these enhancements obviously make the simulator run slower, or in other words, more (simulator) instructions need to be executed per simulated instruction. Note also that a simulator actually simulates the behavior of a microprocessor in a sequential way whereas the microprocessor itself executes the computer program in a parallel way by detecting and exploiting instruction-level parallelism.

If we denote the simulation slowdown factor of a simulator S, then the total simulation time T becomes proportional to the following product: $T \sim P \cdot W \cdot I \cdot S$.

From this section, *we conclude that the total simulation time is proportional to the product of four parameters*:

(i) *the number of processor configurations P that need to be evaluated*,
(ii) *the number of benchmarks W in the workload*,
(iii) *the average length I (dynamic instruction count) for each benchmark, and*
(iv) *the simulator slowdown factor S*.

5. Reducing the Total Simulation Time

Obviously, if we want to reduce the total simulation time, there are four possible options, namely reducing P, W, I and S. In this chapter, we discuss how we can reduce all four. This results in a simulation methodology that is more efficient than current practice. First, we identify an interesting region in the microprocessor design space through statistical simulation. Statistical simulation is a fast simulation technique that yields quite accurate power/performance predictions. As such, an interesting region can be identified efficiently. Second, once we have determined this region, we can run detailed, and thus slow architectural simulations. Fortunately, we can reduce the total simulation time by the following four techniques: (i) we propose to reduce the total number of program-input pairs in the workload, (ii) we propose to use reduced input sets, i.e., inputs that result in smaller dynamic instruction counts but similar program behavior, (iii) we propose to apply trace sampling, or the selection of representative slices from a long program run, and (iv) we propose to speed

up the architectural simulator by applying various optimizations. As such, we have reduced all four factors:

- the size of the microprocessor design space P by selecting a region of interest through statistical simulation before proceeding to detailed architectural simulations,
- the size of the workload space W by choosing a limited but representative set of program-input pairs,
- the length of a program run, or the dynamic instruction count I of a program run by considering reduced input sets and trace sampling,
- the slowdown factor of the architectural simulator S by optimizing simulators.

The result of this simulation methodology is that the total architectural simulation time and cost can be reduced compared to current practice with several orders of magnitude. These four simulation reduction approaches will be described and discussed in Sections 6, 7, 8 and 9, respectively.

6. Reducing the Design Space

As stated in Section 4.1, the design space that needs to be explored is huge. Design space explorations through enumeration, i.e., by evaluating the performance of each individual design point through detailed architectural simulations, is obviously infeasible. As such, we need methods computer architects can use to guide their design process. These methods should be able to quickly identify a region in the design space with interesting characteristics. This smaller region can then be further evaluated using more detailed, and thus slower, architectural simulations.

In the following subsection, we briefly discuss a recently introduced technique, namely statistical simulation. For a more elaborate discussion on statistical simulation, we refer the interested reader to the PhD thesis by Eeckhout [21]. Section 6.2 discusses analytical modeling techniques. In Section 6.3, we enumerate a number of other approaches that have been reported in the literature.

6.1 Statistical Simulation

The statistical simulation methodology consists of four steps (Fig. 2): program trace generation, statistical profiling, synthetic trace generation and trace-driven simulation. Section 3 discussed how program trace files are generated by means of functional simulation. This section deals with the other three steps of the statistical simulation methodology. First, we discuss statistical profiling. Subsequently, we detail

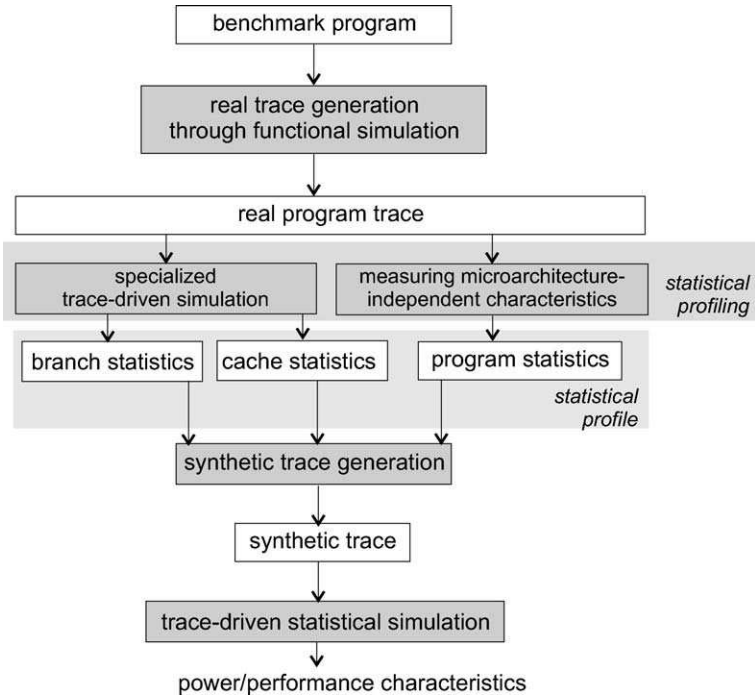


FIG. 2. Statistical simulation: framework.

on the synthetic trace generation algorithm and the trace-driven simulation method. Third, we evaluate the accuracy and the applicability of this methodology. Finally, we will discuss related work concerning statistical simulation.

6.1.1 Statistical Profiling

During the statistical profiling step, a real program trace—i.e., a stream of instructions as they are executed instruction per instruction by a (single-issue in-order) microprocessor—is analyzed by a *microarchitecture-independent* profiling tool and specialized cache and branch predictor simulation tools. The complete set of statistics collected during statistical profiling is called a *statistical profile*. The microarchitecture-independent profiling tool extracts (i) a distribution of the instruction mix (we identify 19 instruction classes according to their semantics and the number of source registers), (ii) the distribution of the age of the input register instances (i.e., the number of dynamic instructions between writing and reading a register instance; measured per instruction class and per source register; 22 distributions

in total) and (iii) the age of memory instances (i.e., the number of load instructions between writing and reading the same memory location). The age distribution of register and memory instances only captures read-after-write (RAW) dependencies. Write-after-write (WAW) and write-after-read (WAR) dependencies are not considered since we assume perfect (hardware supported) register renaming, i.e., there are enough physical registers to remove all WAW and WAR dependencies dynamically. Note that this is not unrealistic for contemporary out-of-order architectures since it is implemented in the Alpha 21 264, see [46]. If measuring performance as a function of the number of physical registers would be needed, the methodology presented here can be easily extended for this purpose by modeling WAR and WAW dependencies as well.

The specialized cache and branch predictor simulators only extract statistics concerning the branch and cache behavior of the program trace for a specific branch predictor and a specific cache organization. The *branch statistics* consist of seven probabilities: (i) the conditional branch target prediction accuracy, (ii) the conditional branch (taken/not-taken) prediction accuracy, (iii) the relative branch target prediction accuracy, (iv) the relative call target prediction accuracy, (v) the indirect jump target prediction accuracy, (vi) the indirect call target prediction accuracy and (vii) the return target prediction accuracy. The reason to distinguish between these seven probabilities is that the prediction accuracies greatly vary among the various branch classes. In addition, the penalties introduced by these are completely different, see [37]. A misprediction in cases (i), (iii) and (iv) only introduces a few pipeline bubbles in the pipeline. In case of a simple pipeline, this can even be a single-cycle bubble. Cases (ii), (v)–(vii) on the other hand, will cause the entire processor pipeline to be flushed and to be refilled when the mispredicted branch is executed.

The *cache statistics* include two sets of distributions: the data cache and the instruction cache statistics. The data cache statistics contain two probabilities for a load operation, namely (i) the probability that a load needs to access the level-2 (L2) cache—as a result of a level-1 (L1) cache miss—and (ii) the probability that main memory—as a result of a level-2 (L2) cache miss—needs to be accessed to get its data; idem for the instruction cache statistics.

A statistical profile can be computed from an actual trace but it is more convenient to compute it on-the-fly using a specialized functional simulator, or using an instrumented version of the benchmark program running on a real system. Both approaches eliminate the need to store huge traces. A second note is that although computing a statistical profile might take a long time, it should be done only once for each benchmark with a given input. And since statistical simulation is a fast analysis technique, computing a statistical profile will be worthwhile. A third important note is that measuring microarchitecture-dependent characteristics such as branch prediction accuracy and cache miss rates, implies that statistical simulation cannot be used

to efficiently study branch predictors or cache organizations. I.e., different statistical profiles need to be computed for different branch predictors and different cache organizations. However, we believe this is not a major limitation since, e.g., cache miss rates for various cache sizes can be computed simultaneously using the cheetah simulator by Sugumar and Abraham [82]. Other microarchitectural parameters on the other hand, can be varied freely. Examples of these microarchitectural parameters are the window size, the fetch width, the dispatch width, the number of functional units, the reorder width, the instruction execution latencies, the number of pipeline stages, etc.

6.1.2 Synthetic Trace Generation and Simulation

Once a statistical profile is computed, a *synthetic trace* is generated by a synthetic trace generator. This is based on a Monte Carlo method: a random number is generated between 0 and 1, that will determine a program characteristic using a cumulative distribution function, see Fig. 3(a).

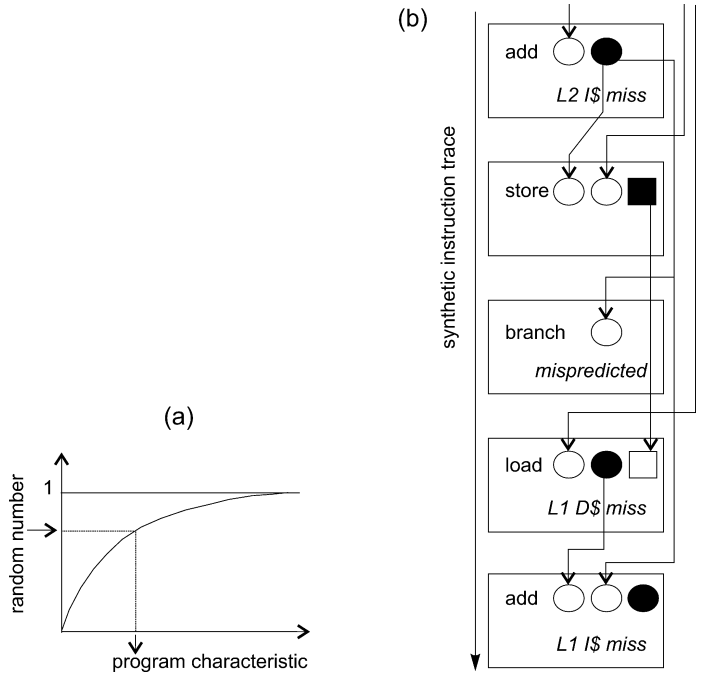


FIG. 3. (a) determining a program characteristic using random number generation and (b) generating a synthetic trace.

The generation of a synthetic trace itself works on an instruction-by-instruction basis. Consider the generation of instruction x in the synthetic instruction stream, see Fig. 3(b):

(1) Determine the instruction type using the instruction-mix distribution; e.g., an add, a store, etc. were generated in Fig. 3(b).
(2) For each source operand, determine the instruction that creates this register instance using the age of register instances distribution. Notice that when a dependency is created in this step, the demand of syntactical correctness does not allow us to assign a destination operand to a store and a conditional branch instruction.[1] For example in Fig. 3(b), the load instruction cannot be made dependent on the preceding branch. However, using the Monte Carlo method we cannot assure that the instruction that is the creator of that register instance, is neither a store nor a conditional branch instruction. This problem is solved as follows: look for another creator instruction until we get one that is not a store nor a conditional branch. If after a certain maximum number of trials still no dependency is found that is not supposedly created by a store or a conditional branch instruction, the dependency is simply removed.
(3) If instruction x is a load instruction, use the age of memory instances distribution to determine whether a store instruction w (before instruction x in the trace; i.e., $w < x$) accesses the same memory address; e.g., a read-after-write dependency is imposed through memory between the load and the store in Fig. 3(b). This will have its repercussions when simulating these instructions. In our simulator we assume speculative out-of-order execution of memory operations. This means that when a load x that accesses the same memory location as a previous store w ($w < x$), is executed earlier than the store, the load would get the wrong data. To prevent this, a table is kept in the processor to keep track of memory dependencies. When the store w is executed later, it will detect in that table that load x has accessed the same memory location. In that case, the load and all its dependent instructions need to be re-executed. A possible implementation of such a table is the Address Resolution Buffer proposed by Franklin and Sohi [30].
(4) If instruction x is a branch, determine whether the branch and its target will be correctly predicted using the branch statistics. In order to model resource contention due to branch mispredictions, we take the following action while simulating a synthetically generated trace: when a 'mispredicted'-labeled branch is inserted in the processor pipeline, instructions are injected in the pipeline (also synthetically generated) to model the fetching from a misspeculated con-

[1] Relative jumps, indirect jumps and returns do not have destination operands either. However, we will not mention further although we take this into account.

trol flow path. These instructions are then marked as coming from a misspeculated path. When the misspeculated branch is executed, the instructions of the misspeculated path are removed, new instructions are fetched (again synthetically generated) and marked as coming from the correct control flow path.

(5) If instruction x is a load instruction, determine whether the load will cause an L1 cache hit/miss or L2 cache hit/miss using the data cache statistics. When an 'L1 or L2 cache miss'-labeled load instruction is executed in the pipeline, the simulator assigns an execution latency according to the type of the cache miss. In case of an L1 cache miss, the L2 cache access time will be assigned; in case of an L2 cache miss, the memory access time will be assigned.

(6) Determine whether or not instruction x will cause an instruction cache hit/miss at the L1 or L2 level. In Fig. 3(b), the first and the last instruction get the label 'L2 I$ miss' and 'L1 I$ miss', respectively. When a 'L1 or L2 cache miss'-labeled instruction is inserted into the pipeline, the processor will stop inserting new instructions in the pipeline during a number of cycles. This number of cycles is the L2 cache access or the memory access time in case of L1 cache miss or a L2 cache miss, respectively.

The last phase of the statistical simulation method is the trace-driven simulation of the synthetic trace which yields estimates of performance and/or power characteristics. An important performance characteristic is the average number of instructions executed per cycle (IPC) which can be easily calculated by dividing the number of instructions simulated by the number of execution cycles. An important power characteristic is the average energy consumption per cycle (EPC).

6.1.3 Evaluation

In this section, we evaluate the applicability of the statistical simulation methodology for quickly exploring huge design spaces. First of all, we evaluate the absolute accuracy which measures how well statistical simulation estimates real performance in one single design point, see Fig. 4. For example, the instructions executed per cycle (IPC) that is estimated through statistical simulation is compared versus the IPC that is obtained through detailed simulations using real program traces—in this case the IBS traces, see [84]—in the left graph of Fig. 4. The right graph presents the absolute accuracy of statistical simulation concerning the energy that is consumed per cycle (EPC). For the microarchitecture configurations that were used in Fig. 4 the IPC prediction errors are no larger than 12% with an average error of 8%; for the energy per cycle estimates, the maximum error is about 5%. As such, we can conclude that statistical simulation is quite accurate.

Although absolute accuracy is important, we believe that relative accuracy (the relationship between multiple design points) is even more important for the purpose of

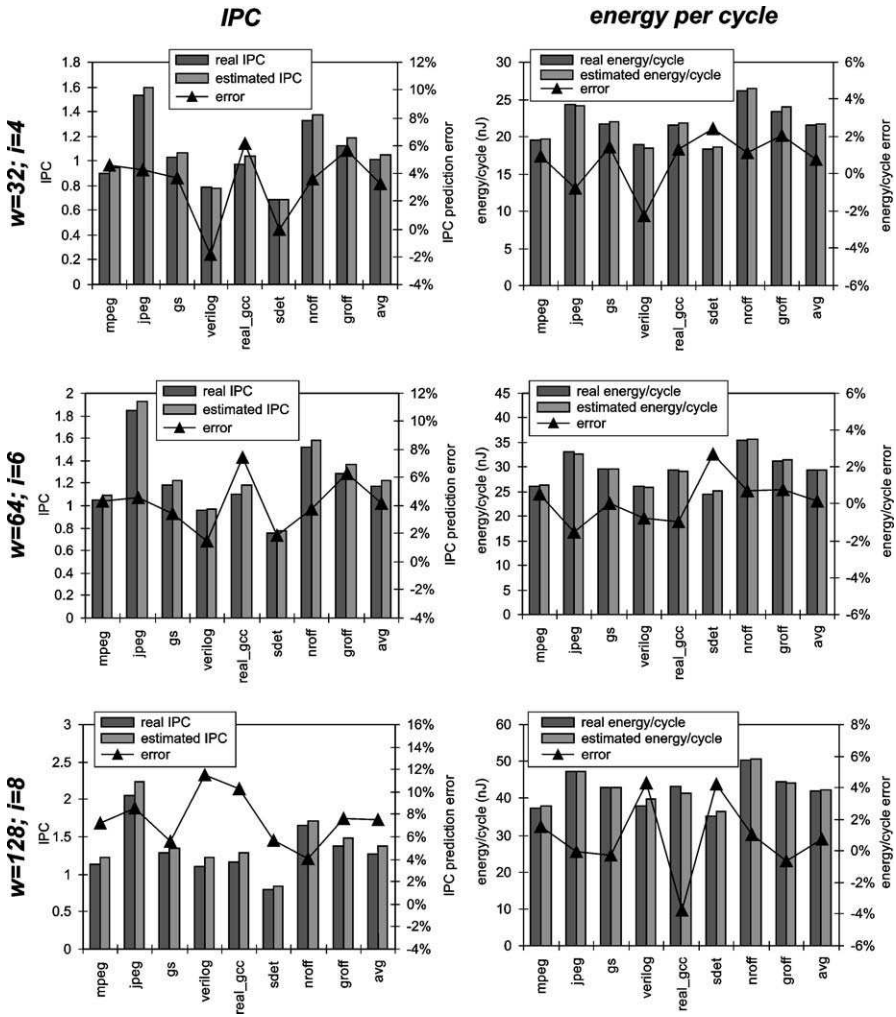


FIG. 4. Absolute accuracy of statistical simulation: instructions executed per cycle or IPC (on the left) and energy consumption per cycle or EPC (on the right). For three superscalar processor configurations: on the top, window size equal to 32 and an issue width of 4; in the middle, window size of 64 instructions and an issue width of 6; and at the bottom, window size of 128 instructions and an issue width of 8.

design space explorations. I.e., when computer architects can make use of accurate estimations of performance trends as a function of microarchitectural parameters, appropriate design decisions can be based upon them. For example, when the per-

formance gain due to increasing a particular hardware resource does not justify the increased hardware cost, designers will decide not to increase that hardware resource. Let us now clarify the relationship between absolute and relative accuracy:

- obviously, perfect absolute accuracy implies perfect relative accuracy;
- an absolute prediction accuracy that is constant over the complete design space still implies a perfect relative accuracy;
- an absolute prediction accuracy that is not constant over the design space might lead to a relative prediction error. However, if the absolute accuracy does not vary wildly over the design space, we can still expect to achieve reasonable relative accuracy.

In case of statistical simulation, the absolute accuracy is not constant over the design space. As such, a relative prediction error might occur. However, the following experiments show that the relative prediction errors for statistical simulation are small and that these small prediction errors do not hold us back from taking correct design decisions.

In Fig. 5, the 'real' and the 'estimated' energy-delay product (EDP) is shown as a function of issue width (along the Y-axis) and window size (along the X-axis). The energy-delay product (EDP) is a fused metric (combining performance as well as energy consumption in one single metric) that is commonly used to evaluate the energy-efficiency of general-purpose microprocessors, see [9]. The most energy-efficient architecture is the one with the lowest energy-delay product, thus maximizing performance with a reasonable energy consumption per cycle. The 'real' EDP numbers, displayed in the left graph of Fig. 5, identify configuration (*window size* = 48 and *issue width* = 6) as the most energy-efficient microarchitecture. These EDP numbers are obtained through real trace simulation. The same configuration is identified by

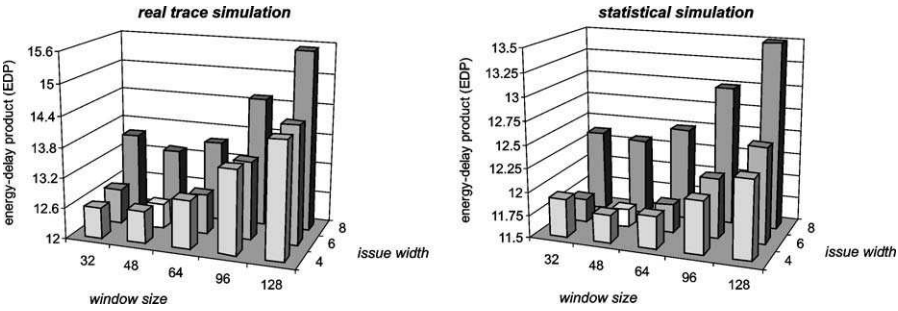


FIG. 5. Relative accuracy of statistical simulation for superscalar processors: real EDP (on the left) versus estimated EDP (on the right) as a function of processor window size and issue width. The optimal configuration, i.e., with the lowest EDP, is shown through a white bar.

statistical simulation, see the right graph of Fig. 5. Note that although the prediction error of the energy-delay product is not constant over the design space—it increases for processors with larger resources—a correct design decision is still made. Although statistical simulation was capable in this experiment to identify *the* most energy-efficient architecture, note that it cannot be guaranteed that statistical simulation will always identify the optimal design due to the fact that the absolute accuracy is not constant over the design space. However, from this experiment we can conclude with confidence that statistical simulation is capable of accurately identifying a region of interest in a huge design space.

Concerning the evaluation speed of statistical simulation we can state that statistical simulation is a very fast simulation technique. From various experiments as described in [21], we have concluded that steady state power/performance characteristics are obtained after simulating only a few hundred thousands or at most 1 million synthetically generated instructions. The simulation time of 1 million instructions is in the order of seconds. If we compare this to the simulation time of real program traces that require the simulation of several billions of instructions, we can conclude that statistical simulation indeed is a very fast simulation technique yielding a speedup of at least a factor one thousand. As such, *given its accuracy and its speed, we can conclude that statistical simulation indeed is a useful technique to quickly explore huge microprocessor design spaces.*

6.1.4 Related Work on Statistical Simulation

Noonburg and Shen [63] presented a framework that models the execution of a program on a particular architecture as a Markov chain, in which the state space is determined by the microarchitecture and in which the transition probabilities between the various states are determined by the program execution. Noonburg and Shen report (while considering perfect caches) a maximum IPC prediction error of 1% for a single-pipeline in-order processor and a maximum IPC prediction error of 10% for a three-pipe in-order processor. This approach has the major disadvantage of becoming too complex in case of wide-resource superscalar processors.

Hsien and Pedram [40] present a technique to estimate performance and power dissipation of a microprocessor by first measuring a characteristic profile of a program execution, and by subsequently synthesizing a new, fully functional program that matches the extracted characteristic profile. The characteristic profile includes the instruction mix, branch prediction accuracy, cache miss rate, pipeline stall rate and IPC. The program that is synthesized using this characteristic profile, is then executed on an execution-driven simulator to estimate performance and power consumption. Since the dynamic instruction count of the synthesized program is smaller than the dynamic instruction count of the original program, the simulation time is significantly reduced. The prediction errors for both power dissipation and IPC are less

than 5%. The major drawback of this approach is that no distinction is made between microarchitecture-independent and microarchitecture-dependent characteristics; all characteristics are microarchitecture-dependent. Consequently, this approach cannot be used for architectural design space explorations.

The statistical simulation methodology as it is discussed here, was initially presented by Carl and Smith [12]. They proposed an approach in which a synthetic instruction trace is generated based on execution statistics and is subsequently fed into a trace-driven simulator. Nussbaum and Smith [64] continued this work and presented a different method for generating inter-operation dependencies. We generate what they call *upstream dependencies*, i.e., an instruction is made dependent on a preceding instruction. Nussbaum and Smith on the other hand, use so called *downstream dependencies*, which means that a future instruction is made dependent on the current instruction. Nussbaum and Smith also present an evaluation of using various higher-order distributions in which the instruction mix, the inter-operation dependencies, the cache miss rates and the branch misprediction rates are correlated to the basic block size. The authors conclude that these higher-order distributions indeed can lead to higher performance prediction accuracies, e.g., the average IPC prediction error can be reduced from 15% to 9% for a wide-resource microprocessor configuration. However, they also report experimental results suggesting that simple statistical models are accurate enough for doing design space explorations.

Nussbaum and Smith [65] continued their work by evaluating symmetric multiprocessor system (SMP) performance through statistical simulation. They evaluated multiprogrammed workloads as well as parallel scientific workloads and conclude that statistical simulation is sufficiently accurate to predict SMP performance trends.

Oskin et al. [68] present the HLS simulation environment which is basically the same as the statistical simulation methodology presented by Carl and Smith [12] and the model presented here. The work done by Oskin et al. [68] has two major contributions. First, they validate the statistical simulation methodology against real hardware, namely a MIPS R10000 processor, see Yeager [87], and they conclude that statistical simulation indeed achieves a high performance prediction accuracy (a maximum error of 7.8% is reported). Second, they evaluate how well statistical simulation predicts performance under varying branch prediction accuracies, L1 I-cache miss rates, L1 D-cache miss rates and compiler optimization levels. These experiments are so called single-value correlation studies, i.e., by varying only one parameter in each experiment. They also performed multi-value correlation studies by varying several parameters simultaneously. This kind of experiments is extremely useful for identifying in which area of the design space statistical simulation can be used with confidence.

6.2 Analytical Modeling

Instead of, or in addition to the statistical simulation methodology, analytical modeling can also be used for quickly exploring the design space. However, to the best of our knowledge, there exists no analytical model that is as accurate and as flexible as statistical simulation. Note that statistical simulation attains accurate performance predictions and is extremely simple to implement: measuring the statistical profile is straightforward as well as the generation of a synthetic trace. In addition, the simulator that is used to simulate the synthetic trace is very simple since only a limited functionality of the microarchitecture needs to be modeled. An analytical model on the other hand, is harder to develop because of the complex behavior of contemporary microarchitectures. As such, analytical models typically assume several assumptions to be true, such as unit execution latencies of instructions, unlimited number of functional units or only one functional unit, memory dependencies not being modeled, etc. In statistical simulation on the other hand, these assumptions do not need to be fulfilled.

Noonburg and Shen [62] present an analytical model of the interaction between program parallelism and machine parallelism. This is done by combining component functions concerning the program (modeling data and control parallelism) as well as the machine (modeling branch, fetch and issue parallelism). The program characteristics are measured only once for each benchmark as it is the case for statistical simulation. The machine characteristics are obtained by analyzing the microarchitecture. This analytical model attains quite accurate performance predictions. Unfortunately, unit instruction execution latencies are assumed and memory dependencies are not modeled.

Dubey et al. [18] propose an analytical performance model on the basis of two parameters that are extracted from a program trace. The first parameter, the *conditional independence probability* p_δ, is defined as the probability that an instruction x is independent of instruction $x - \delta$ given that x is independent of all instructions in the trace between x and $x - \delta$. The second parameter p_ω is defined as the probability that an instruction is scheduled with an instruction from ω basic blocks earlier in the program trace. The main disadvantages of this analytical model are that only one dependency is considered per instruction and that no differentiation is made between various instruction classes for p_δ, which will lead to inaccurate performance estimates. In a follow-up research paper, Kamin III et al. [44] propose to approximate the conditional independence probability p_δ using an exponential distribution. Eeckhout and De Bosschere [22] on the other hand, show that a power law is a better approximation than the exponential distribution.

Squillante et al. [79] propose analytical models to capture the workload behavior and to estimate pipeline performance. Their technique was evaluated for a single-issue pipelined processor.

Sorin et al. [78] present a performance analysis methodology for shared-memory systems that combines analytical techniques with traditional simulations to speed up the design process.

Although current analytical models are not flexible enough or accurate enough for performing design space explorations of contemporary superscalar processors, they can be extremely useful to investigate particular parts of a microarchitecture. As such, simple analytical models are often used to get insight in the impact of microarchitectural parameters on performance, see for example [28,29,32,56,60].

6.3 Other Approaches

Conte [13] presents an approach that automatically searches near-optimal processor configurations in a huge design space. His technique is based on simulated annealing. The architectural parameters that are varied in his experiments are the number of functional units, their types and their execution latencies.

Brooks et al. [10] evaluate the popular abstraction via separable components method, see for example Emma [28], which considers performance as the summation of a base performance level (idealized base cycles per instruction or CPI while assuming perfect caches, perfect branch prediction, etc.) plus additional stall factors due to conflicts, hazards, cache misses, branch mispredictions, etc. A simulation speedup is obtained with this technique since the base performance level and the stall factors can be computed using simple simulators instead of fully-detailed and thus slower simulators. They conclude that for modeling out-of-order architectures, this methodology attains, in spite of its poor absolute accuracy, a reasonable relative accuracy.

7. Reducing the Workload

In Section 4.2, we argued that the selection of a representative workload is extremely important throughout the entire microprocessor design flow. A representative workload consists of a selected number of benchmarks with well chosen inputs that are representative for the target domain of operation of the microprocessor currently under design. A naive approach to the workload composition problem would be to select a huge number of benchmarks and for each benchmark a large number of inputs. Since the total simulation time is proportional to the number of program-input pairs in the workload, this approach is infeasible. As such, we propose to chose a selected number of program-input pairs that cover the workload space with confidence, see [26].

Conceptually, the complete workload design space can be viewed as a p-dimensional space with p the number of important program characteristics that affect performance, e.g., branch prediction accuracy, cache miss rates, instruction-level parallelism (ILP), etc. Obviously, p will be too large to display the workload design space understandably. In addition, correlation exists between these variables which reduces its understandability. As such, it is hard to determine which program characteristics make the diversity in the workload space. In our methodology, we reduce the p-dimensional workload space to a q-dimensional space with $q \ll p$ ($q = 2$ to $q = 4$ typically) making the visualisation of the workload space possible without losing important information. This is achieved by using statistical data reduction techniques such as principal components analysis (PCA) and cluster analysis (CA).

Each benchmark-input pair is a point in this (reduced) q-dimensional space obtained after PCA. We can expect that different benchmarks will be 'far away' from each other while different input data sets for a single benchmark will be clustered together. This representation gives us an excellent opportunity to measure the impact of input data sets on program behavior. Weak clustering for various inputs and a single benchmark indicates that the input set has a large impact on program behavior; strong clustering on the other hand, indicates a small impact. This representation can be helpfully used during workload design. Indeed, strong clustering suggests that a single or only a few input sets should be selected as a representative for the cluster. This will reduce the total simulation time significantly since the total number of benchmark-input pairs in the workload is reduced.

Before going into detail on the results that can be obtained using this methodology, we will first give a brief explanation of the multivariate statistical data analysis techniques that we have used. We first discuss principal components analysis. Subsequently, we explain how cluster analysis works. For a more detailed description of both techniques, we refer the interested reader to [53].

7.1 Principal Components Analysis

Principal components analysis (PCA) is a statistical data analysis technique that presents a different view on the measured data. It builds on the assumption that many variables (in our case, workload characteristics) are correlated and hence, they measure the same or similar properties of the program-input pairs. PCA computes new variables, called *principal components*, that are *linear combinations* of the original variables, such that all principal components are uncorrelated. In other words, PCA transforms the p variables $X_1, X_2, \ldots, X_p$ into p principal components $Z_1, Z_2, \ldots, Z_p$ with $Z_i = \sum_{j=1}^{p} a_{ij} X_j$. This transformation has the properties (i) $\text{Var}[Z_1] > \text{Var}[Z_2] > \cdots > \text{Var}[Z_p]$ which means that Z_1 contains the most information and Z_p the least; and (ii) $\text{Cov}[Z_i, Z_j] = 0,\ \forall i \neq j$ which means that

there is no information overlap between the principal components. Note that the total variance in the data remains the same before and after the transformation, namely $\sum_{i=1}^{p} \text{Var}[X_i] = \sum_{i=1}^{p} \text{Var}[Z_i]$.

As stated in the first property in the previous paragraph, some of the principal components will have a high variance while others will have a small variance. By removing the components with the lowest variance from the analysis, we can reduce the number of workload characteristics while controlling the amount of information that is thrown away. We retain q principal components which is a significant information reduction since $q \ll p$ in most cases, typically $q = 2$ to $q = 4$. To measure the fraction of information retained in this q-dimensional space, we use the amount of variance $(\sum_{i=1}^{q} \text{Var}[Z_i])/(\sum_{i=1}^{p} \text{Var}[X_i])$ accounted for by these q principal components.

In this study the p original variables are the workload characteristics. By examining the most important q principal components, which are linear combinations of the original workload characteristics, meaningful interpretations can be given to these principal components in terms of the original workload characteristics. A coefficient a_{ij} that is close to $+1$ or -1 implies a strong impact of the original characteristic X_j on the principal component Z_i. A coefficient A_{ij} that is close to zero on the other hand, implies no impact.

The next step in the analysis is to display the various benchmarks as points in the q-dimensional space built up by the q principal components. This can be done by computing the values of the q retained principal components for each program-input pair. This representation gives us an excellent opportunity to visualize the workload design space understandably. Note that this q-dimensional space will be much easier to understand than the original p-dimensional space for two reasons: (i) q is much smaller than p and (ii) the q-dimensional space is uncorrelated.

During principal components analysis, one can either work with normalized or non-normalized data (the data is normalized when the mean of each variable is zero and its variance is one). In the case of non-normalized data, a higher weight is given in the analysis to variables with a higher variance. In our experiments, we have used normalized data because of our heterogeneous data; e.g., the variance of the instruction-level parallelism (ILP) is orders of magnitude larger than the variance of the data cache miss rates.

7.2 Cluster Analysis

Cluster analysis (CA) is a data analysis technique that is aimed at clustering n cases, in our case program-input pairs, based on the measurements of q variables, in our case the principal components as obtained from PCA. The final goal is to obtain a number of clusters, containing program-input pairs that have 'similar' behavior.

There exist two commonly used types of clustering techniques, namely linkage clustering and K-means clustering.

Linkage clustering starts with a matrix of distances between the n cases or program-input pairs. As a starting point for the algorithm, each program-input pair is considered as a cluster. In each iteration of the algorithm, the two clusters that are most close to each other (with the smallest distance, also called the *linkage distance*) will be combined to form a new cluster. As such, close clusters are gradually merged until finally all cases will be in a single cluster. This can be represented in a so called *dendrogram*, which graphically represents the linkage distance at each iteration of the algorithm. Having obtained a dendrogram, it is up to the user to decide how many clusters to take. This decision can be made based on the linkage distance. Indeed, small linkage distances imply strong clustering and thus high similarity, while large linkage distances imply weak clustering or dissimilar behavior.

Cluster analysis is heavily dependent on an appropriate distance measure. In our analysis, the distance between two program-input pairs is computed as the Euclidean distance in the transformed q-dimensional space obtained after PCA for the following reason. The values along the axes in this space are uncorrelated. The absence of correlation is important when calculating the Euclidean distance because two correlated variables—that essentially measure the same thing—would contribute a similar amount to the overall distance as an independent variable; as such, these variables would be counted twice, which is undesirable.

Next to defining the distance between two program-input pairs, we also need to define the distance between two clusters of program-input pairs. One way to compute the distance between two clusters is the *furthest neighbor* strategy or *complete linkage*. This means that the distance between two clusters is defined as the largest distance between two members of each cluster. Another possibility is the *weighted pair-group average* strategy in which the distance between two clusters is computed as the weighted average distance between all the pairs of program-input pairs in the two different clusters. The weighting of the average is done by considering the cluster size, i.e., the number of program-input pairs in the cluster.

The second type of clustering techniques is *K-means clustering*. K-means clustering produces exactly K clusters with the greatest possible distinction. The algorithm works as follows. In each iteration, the distance is calculated for each program-input pair to the each cluster center. A program-input pair then gets assigned to the nearest cluster. As such, the new cluster centers can be computed. This algorithm is iterated until no more changes are observed in the cluster membership.

7.3 Evaluation

In [25], we have shown how to reduce a collection of 79 program-input pairs taken from the SPEC CPU95 integer benchmark suite[2] and the TPC-D queries,[3] to only 16 program-input pairs. As such, the total simulation time is reduced by nearly a factor of 5.

The workload characteristics that were used in this analysis are shown in Table I. These characteristics are commonly used in the computer architecture literature to characterize general-purpose applications. Measuring these program characteristics

TABLE I
THE WORKLOAD CHARACTERISTICS USED TO REDUCE THE WORKLOAD

Category	No.	Description
Instruction mix	1	Percentage integer arithmetic operations
	2	Percentage logical operations
	3	Percentage shift and byte manipulation operations
	4	Percentage load and store operations
	5	Percentage control transfer operations (branches, etc.)
Branch prediction behavior	6	Branch prediction accuracy of 16 Kbit bimodal branch predictor
	7	Branch prediction accuracy of 16 Kbit gshare branch predictor
	8	Branch prediction accuracy of 48 Kbit hybrid branch predictor consisting of a bimodal and a gshare component
Data cache behavior	9	Cache miss rate of 8 KB direct-mapped cache
	10	Cache miss rate of 16 KB direct-mapped cache
	11	Cache miss rate of 32 KB 2-way set-associative cache
	12	Cache miss rate of 64 KB 2-way set-associative cache
	13	Cache miss rate of 128 KB 4-way set-associative cache
Instruction cache behavior	14	Cache miss rate of 8 KB direct-mapped cache
	15	Cache miss rate of 16 KB direct-mapped cache
	16	Cache miss rate of 32 KB 2-way set-associative cache
	17	Cache miss rate of 64 KB 2-way set-associative cache
	18	Cache miss rate of 128 KB 4-way set-associative cache
Control flow	19	Number of instructions between two sequential flow breaks, i.e., number of instructions between two taken branches
Instruction-level parallelism (ILP)	20	Amount of ILP in case of an infinite-resource machine, i.e., infinite number of functional units, perfect caches, perfect branch prediction, etc.

[2]http://www.spec.org.
[3]http://www.tpc.org.

was done using ATOM, a binary instrumentation tool for the Alpha architecture. ATOM allows the instrumentation of functions, basic blocks and individual instructions. For more information on this tool, we refer to [80]. As Table I shows, the total number of characteristics is 20. As such, we have a data matrix of 79 rows (the program-input pairs) and 20 columns (the workload characteristics). We have used STATISTICA by StatSoft, Inc. [81], a statistical data analysis software package, to perform the principal components analysis and the cluster analysis.

The reduced workload space that is obtained after PCA is shown in Fig. 6. The original workload space, which was 20-dimensional, was reduced through PCA to a 4-dimensional space that is visualized in these graphs: the workload space is shown as a function of the first and the second dimension in the top graph, and as a function of the third and the fourth dimension in the bottom graph. These four principal components account for 89.5% of the total variance. The first principal component accounts for 29.7% of the total variance and basically quantifies the amount of (taken) branches and the instruction cache behavior. As such, program-input pairs that are far away from each other along the first principal component will have a completely different percentage (taken) branches and a completely different instruction cache behavior. For example, program-input pairs with a positive value along the first principal component show a relatively small amount of control transfer operations and a small I-cache miss rate. For program-input pairs with a negative value along the first principal component, the reverse is true. In conclusion, we can state that the first principal component somehow quantifies the locality in the instruction stream. The second principal component accounts for 28.0% of the total variance and measures the amount ILP, branch behavior and percentage logical operations. The third component accounts for 18.5% of the total variance and quantifies the amount of arithmetic operations as well as the data cache behavior. The fourth component account for 13.3% of the total variance and measures the amount of shift and load/store operations.

As explained previously, the next step in the workload design methodology is to apply cluster analysis. The dendrogram that is obtained from (linkage distance based) cluster analysis is shown in Fig. 7. All the 79 program-input pairs included in the analysis are shown along the Y axis of this graph. Program-input pairs that are connected through short linkage distances are close to each other in the workload space and thus exhibit similar program behavior. Program-input pairs that are connected through long linkage distances exhibit dissimilar behavior.

How can these results now be applied during workload design? Since the linkage distance quantifies the (dis)similarity between program-input pairs, we can use this metric to define the reduced workload. For example, if we allow for 16 program-input pairs in our reduced workload, we determine the critical linkage distance. In Fig. 7, this critical linkage distance is slightly greater than 1, as is shown through the

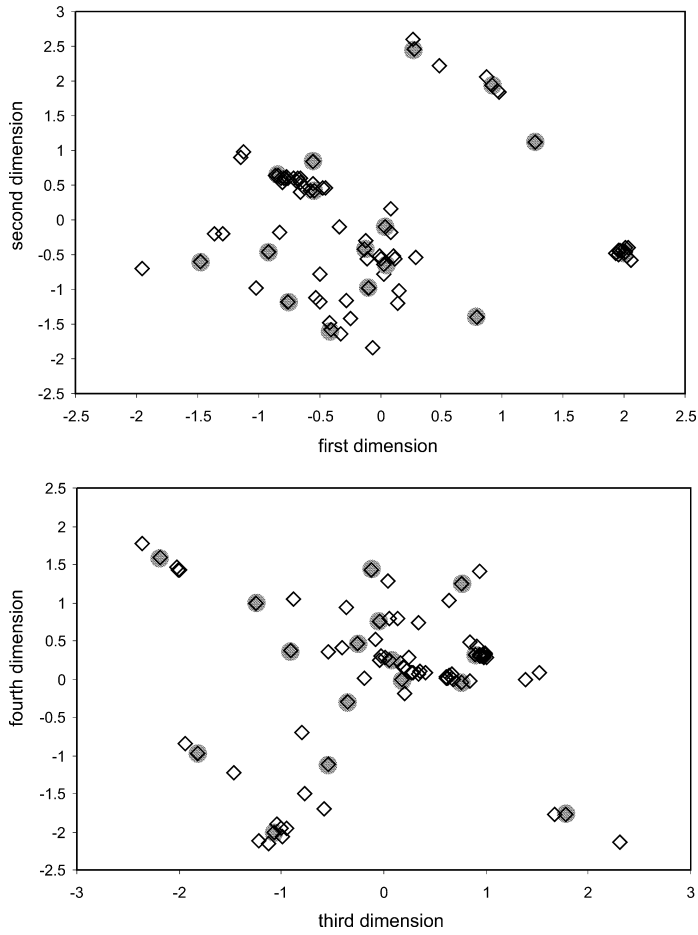


FIG. 6. The transformed workload space obtained after PCA: the first and the second principal component are shown in the top graph; the third and the fourth principal component are shown in the bottom graph. The program-input pairs that are selected for the reduced workload are shown in a gray circle.

vertical dashed line. All the program-input pairs that are connected through linkage distances that are smaller than the critical linkage distance are members of the same cluster. Program-input pairs connected on the other hand, through a linkage distance that is larger than the critical linkage distances are not members of the same cluster (by definition of the critical linkage distance, there will be 16 clusters). Now we can choose a representative program-input pair for each cluster. We have chosen

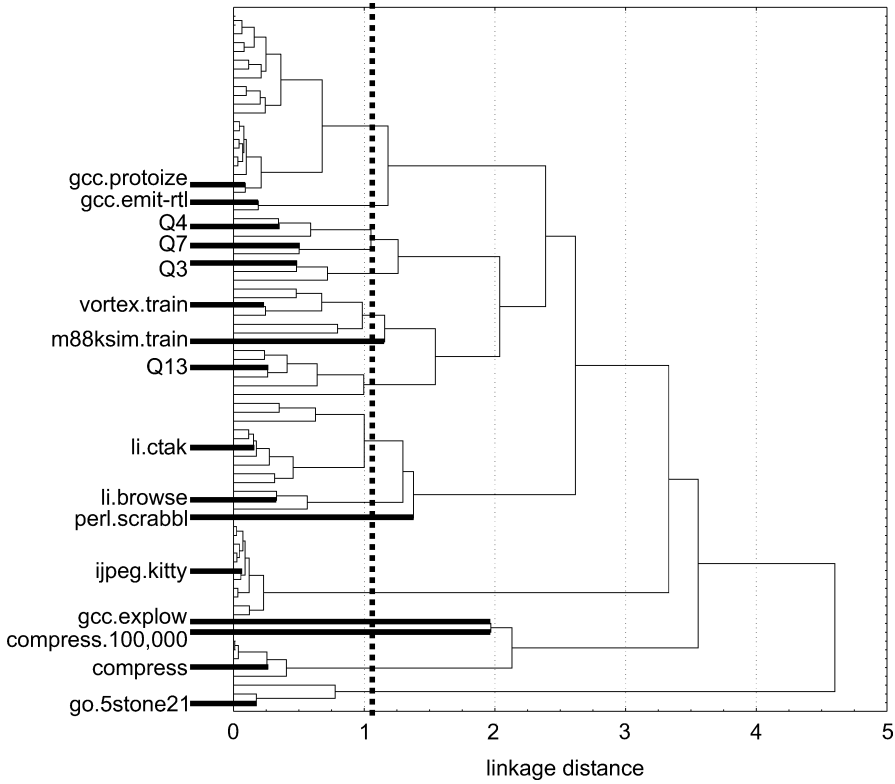


FIG. 7. Selecting representative program-input pairs from a dendrogram.

the one with the smallest dynamic instruction count *and* being close to the center of the cluster it belongs to. As a result, these representative program-input pairs now constitute the reduced workload. The program-input pairs that were selected from our analysis are shown in Fig. 7 with the bold lines and Fig. 6 as gray circles. Note that the program-input pairs that are selected are more or less uniformly spread over this workload space. Using this reduced workload instead of the original workload results in a simulation speedup of a factor 6—the total dynamic instruction count of the reduced workload is 101 billion instructions whereas the original workload counts 593 billion dynamic instructions.

Recall that next to linkage based clustering, there exists another clustering technique which is called K-means clustering. In case we want to select 16 clusters, we define $K = 16$. By applying K-means clustering, we obtain a classification that is

quite consistent with the one obtained through linkage based clustering as discussed above. The 16 representative benchmark-input pairs are postgres running queries Q4, Q13, Q3, Q7 and Q8 next to perl.primes, gcc.protoize, gcc.print-tree, gcc.explow, li.ctak, li.browse, compress.1,000,000, compress.100,000, ijpeg.kitty, go.5stone21 and m88ksim.train.

As such, from this section *we can conclude that this methodology which is based on principal components analysis and cluster analysis, is capable of selecting a reduced set of representative program-input pairs that spans the workload space reliably.*

7.4 Related Work

The methodology that is presented in the previous sections is built on the idea of measuring benchmark (dis)similarity. Indeed, program-input pairs that are close to each other in the workload space are similar. Program-input pairs that are far away from each other exhibit dissimilar behavior. In the literature there exist only a few approaches to measuring benchmark similarity. Saavedra and Smith [72] present a metric that is based on dynamic program characteristics for the Fortran language, for example the instruction mix, the number of function calls, the number of address computations, etc. For measuring the difference between benchmarks they used the squared Euclidean distance. Our methodology differs from the one presented by Saavedra and Smith [72] for two reasons. First, the program characteristics used in our methodology are more suited for performance prediction of contemporary architectures since we include branch prediction accuracy, cache miss rates, ILP, etc. Second, we prefer to work with uncorrelated program characteristics (obtained after PCA) for quantifying differences between program-input pairs, as extensively argued above.

Yi et al. [88] propose a technique for classifying benchmarks with similar behavior, i.e., by grouping benchmarks that stress the same processor components to similar degrees. Their method is based on a Plackett–Burman design. A Plackett–Burman design is a technique that allows researchers to measure the impact of variables by making a limited number of measurements. For example, consider the case where we want to measure the impact of n variables where each variable can have b unique values. The total number of experiments (or in our case simulations) that need to be done is b^n. This is also called a full multifactorial design. A Plackett–Burman design on the other hand is a fractional multifactorial design. It is a well established technique for measuring the impact of n variables and their interactions by doing a limited number of experiments, namely $2(n + 1)$. This is done by varying all parameters simultaneously in a well chosen 'foldover' design.

8. Reducing the Length of a Program Simulation Run

As discussed in Section 4.3, the total simulation time is also proportional to the dynamic instruction count of a program-input pair. Recall there are two reasons for this phenomenon: (i) the increasing complexity of current computer applications, and (ii) the increasing performance of computer systems. Both force computer architects to consider huge numbers of instructions during architectural simulations. Several tens or even hundreds of billions of instructions are not exceptional nowadays. In this section, we discuss two approaches to this problem. In the first subsection, we detail on trace sampling which is a technique that is well known in the domain of computer architecture for many years now. In the second subsection, we will present and discuss a recently proposed approach, namely reduced input sets. The final subsection will compare both approaches and point out the advantages and disadvantages for each of them.

8.1 Trace Sampling

A *sampled trace* is obtained from an original program trace by gathering *trace samples* from the original trace, the black boxes in Fig. 8—the other details of this figure will be discussed later on. Since a sampled trace is to be used to estimate the performance of the original trace, it is very important how to select the trace samples. In the literature, researchers refer to *representative* trace samples when the sampled trace exhibits similar behavior as the original program trace. Clearly, there are a number of issues that need to be dealt with when selecting representative trace samples. First, it is unclear how many instructions should be contained in a trace sample. The trace samples that are used by researchers vary widely between 50,000 to 300 million instructions. Second, it is not obvious how many of these samples need to be considered in sampled traces. Some researchers report a single sample of 50 million or 300 million instructions. Others take multiple trace samples with a total sample rate of for example 5%, or the total number of instructions in the sampled

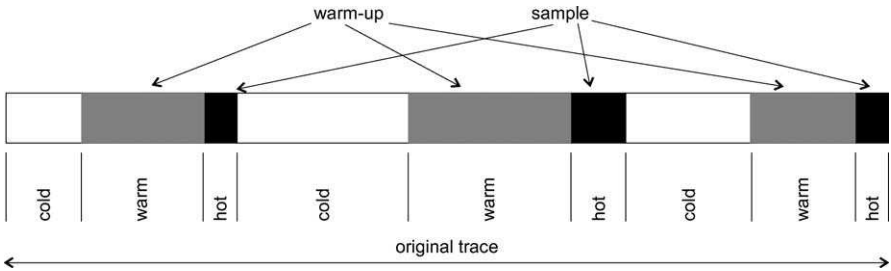


FIG. 8. Trace sampling: basic principles.

trace being limited to for example 1 billion instructions. Third, several methods exist to select these trace samples: (pseudo-)random selection, periodic selection, manual selection, selection through cluster analysis based on some criterion, etc. However, it is unclear which criterion yields the most representative sampled trace. Fourth, a correct hardware state at the beginning of each sample needs to be guaranteed as much as possible. This problem is well known in the literature as the cold-start problem. These four issues will be discussed in the following subsections.

8.1.1 Size and Number of Trace Samples

Different authors have used different trace sample lengths L as well as different numbers of trace samples N per benchmark. Table II gives an overview of values for L and N as observed in the literature. This table shows that the number of trace samples per benchmark varies from 1 to 10,000. The trace sample length varies from 1000 to 100 million instructions. Note that in general a small number of samples coexists with a large sample length and vice versa. For example, Sherwood et al. [75] use 1 to 10 samples of 100 million instructions each. Wunderlich et al.[86] on the other hand, use 10,000 samples of 1000 instructions each. The total length of the sampled traces varies between 240,000 and 1 billion instructions, see rightmost column in Table II.

To measure the impact of the size of the trace samples as well as the number of trace samples on the accuracy of the sampled trace, we have set up the following experiment. We have measured the data cache miss rates for each interval of 1 million

TABLE II
THE NUMBER OF TRACE SAMPLES PER BENCHMARK, THE TRACE SAMPLE LENGTH AND THE TOTAL LENGTH OF THE SAMPLED TRACE AS OBSERVED IN THE LITERATURE

Paper	Number of samples N	Sample length L	Total length
[76]	1	50,000,000	50,000,000
[75]	1 to 10	100,000,000	100,000,000 to 1,000,000,000
[85]	19 to 35	10,000 to 5,000,000	19,000 to 175,000,000
[49]	35	60,000	2,100,000
[14]	40	100,000	4,000,000
[35]	50	1,000,000	50,000,000
[61]	8 to 64	30,000 to 380,000	240,000 to 23,520,000
[50]	at least 50	250,000	at least 12,500,000
[54]	10 to 100	500,000	5,000,000 to 50,000,000
[45]	20 to 200	100,000 to 1,000,000	2,000,000 to 200,000,000
[16]	20 to 300	500,000	10,000,000 to 150,000,000
[15]	2,000	1,000 to 10,000	2,000,000 to 20,000,000
[86]	10,000	1,000	10,000,000

instructions for six SPEC CPU2000 integer benchmarks and six SPEC CPU2000 floating-point benchmarks. This was done using the cheetah cache simulation routines. The actual routines are taken from the SimpleScalar tool set distribution.[4] The cheetah cache simulation routines allow the simulation of multiple cache configurations in parallel. This is achieved by maintaining least-recently used (LRU) stacks of references, see [82]. The instrumentation itself of the Alpha binaries with these cache simulation routines is done using ATOM. For the experiments presented in this section, we instrumented all the memory operations with the cache simulation routines.

Using the above procedure we obtain data cache miss rates for each interval of 1 million instructions, for each benchmark and for each cache configuration. As such, we can calculate the cache miss rates of sampled traces under various sampling conditions and compare them to the miss rates of the original benchmark traces. More in particular, we want to quantify the maximum error for a given sampling scenario. This is illustrated in Fig. 9 for a periodic sample selection mechanism. At the top of the figure, we display the original program trace that is divided in a number of contiguous intervals each containing B instructions. In our experiments, we assumed $B = 1{,}000{,}000$. The three scenarios that are shown in Fig. 9 are obtained by varying the sample length L and the sample rate. The sample length L is measured in units of the interval size B. The sample rate is defined as the number of instructions in the sampled trace versus the total number of instructions in the original trace. The first scenario assumes $L = B$ and a sample rate of 50%. As such, there are two possible sampled traces: (i) a sampled trace containing intervals 1, 3, 5, 7, etc., and (ii) a sampled trace containing intervals 2, 4, 6, 8, etc. The second scenario assumes $L = B$ and a sample rate of 20%. This scenario results in five possible sampled traces: (i) containing intervals 1, 6, 11, etc., (ii) containing intervals 2, 7, 12, etc., . . . , (v) containing intervals 5, 10, 15, etc. The third scenario assumes $L = 2B$ and a sample rate of 50%. The first sampled trace then contains intervals 1, 2, 5, 6, 9, 10, etc. The second sampled trace contains intervals 3, 4, 7, 8, 11, 12, etc. For all the sampled traces obtained from these scenarios we can now calculate the cache miss rates for a number of cache configurations. As such, for each cache configuration and for each benchmark, we obtain 2, 5 and 2 cache miss rates for the three scenarios, respectively. As such, we can calculate the maximum prediction error $E_{\max}$ for a scenario which is defined as the maximum of the cache miss prediction errors E for each of the sampled traces associated with that scenario and for each cache configuration. On its turn, the cache miss prediction error E for one particular trace sampling scenario and one particular cache configuration is defined as follows:

[4] http://www.simplescalar.com.

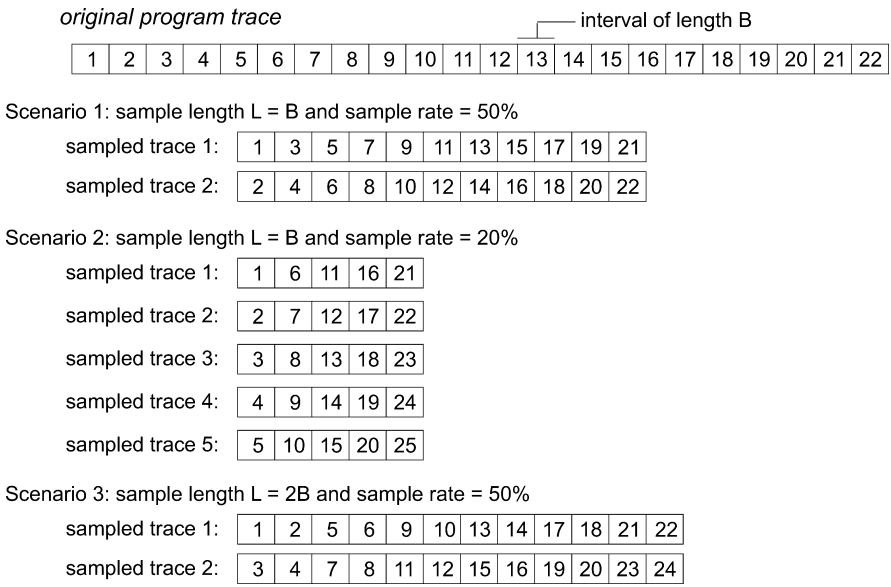


FIG. 9. Different possibilities for choosing sampled traces under a periodic sampling regime for various values of sample length L and sample rate.

$$E = \left| \frac{M_s - M_o}{M_o} \right|, \tag{1}$$

with M_s the cache miss rate for the sampled trace and M_o the cache miss rate for the original trace. For example, for our experiments with the SPEC CPU2000 benchmark vortex, the first scenario results in a maximum prediction error of $E_{max} = 0.12\%$, the second scenario in $E_{max} = 1.06\%$ and the third scenario in $E_{max} = 0.29\%$. Note that in this section, we assume perfect warmup at the beginning of each sample.

For studying the impact of the number of trace samples and their sizes, we have considered eight cache configurations under various trace sampling scenarios. We considered the following cache configurations: a 8 KB direct-mapped cache, a 16 KB direct-mapped cache, a 32 KB 2-way set-associative cache, a 64 KB 2-way set-associative cache, a 128 KB 4-way set-associative, a 256 KB 4-way set-associative cache, a 512 KB 8-way set-associative cache and a 1 MB 8-way set-associative cache. For all these cache configurations, a block size of 32 bytes is assumed. The various trace sampling scenarios are obtained by choosing three values for L, namely 1 million, 10 million and 100 million instructions. The sample rate is varied between 50% and 0.1%. The results for six SPEC CPU2000 integer benchmarks and six SPEC CPU2000 floating-point benchmarks are shown in Figs. 10 and 11, re-

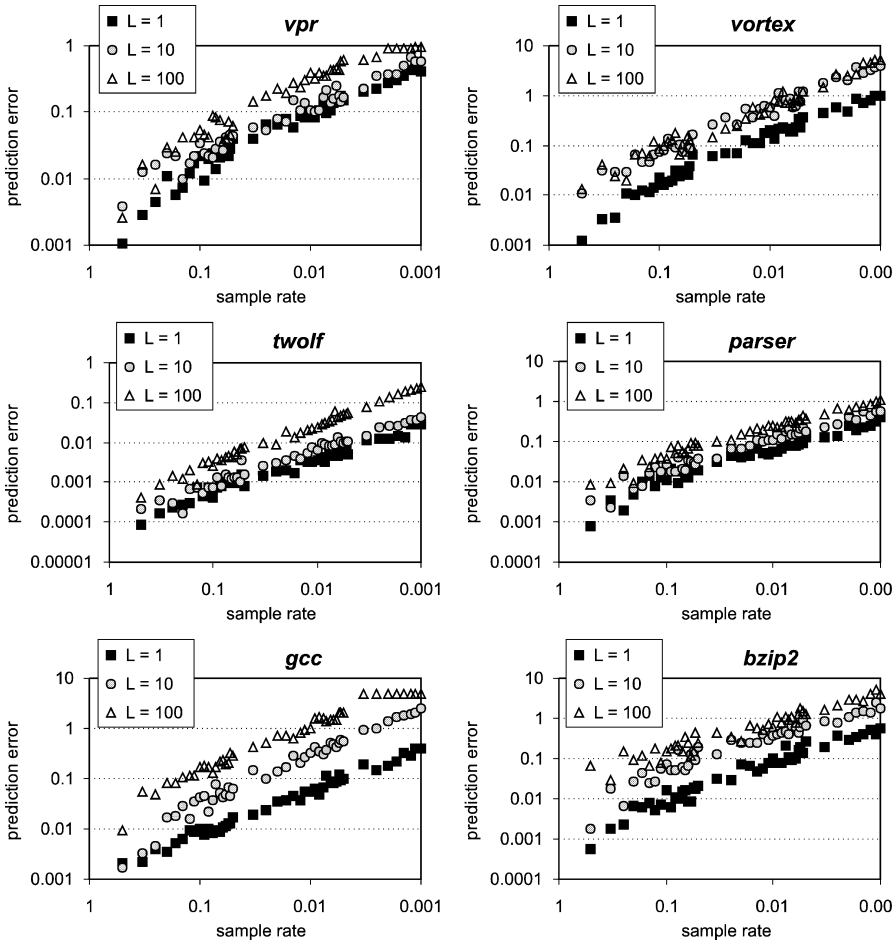


FIG. 10. The maximum data cache miss rate prediction error versus trace sample rate for six (randomly chosen) SPEC CPU2000 integer benchmarks.

spectively. From these figures *we can conclude that small trace samples* (*containing fewer instructions*) *are a better option than large trace samples for the same sample rate.* Indeed, for a given sample rate the maximum prediction error is generally smaller for a sampled trace containing trace samples of 1 million instructions than for a sampled trace containing trace samples of 10 million or 100 million instructions. For example for gcc and a sample rate of 10%, the maximum prediction error is 1%, 4% and 16% for trace samples containing 1 million, 10 million and 100 mil-

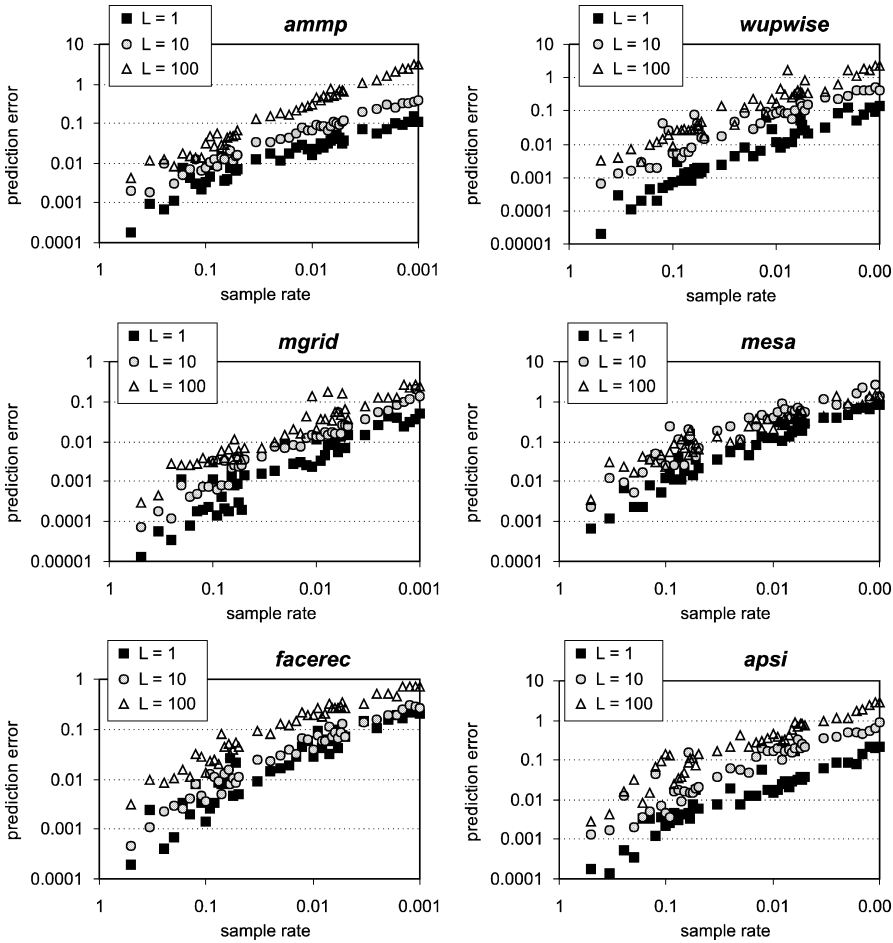


FIG. 11. The maximum data cache miss rate prediction error versus trace sample rate for six (randomly chosen) SPEC CPU2000 floating-point benchmarks.

lion instructions, respectively. *The reverse is also true: a given level of accuracy can be obtained with a much smaller sample rate for sampled traces with small trace samples than with large trace samples.* Again for gcc, a maximum prediction error of 10% is obtained for a sample rate of 0.5% when a trace sample contains 1 million instructions. In case a trace sample contains 10 million instructions, the sample rate should be 2.5% to obtain the same level of performance prediction accuracy. In case 100 million instruction samples, the sample rate should be 14%.

These results can be explained intuitively. Assume two trace sampling scenarios: (i) a first sampled trace consisting of N samples with each sample containing $s \cdot B$ instructions, and (ii) a second sampled trace consisting of $s \cdot N$ samples with each sample containing B instructions. Note that both scenarios have the same sample rate because both sampled traces contain $s \cdot N \cdot B$ instructions. From the literature, we know that the execution of a computer program typically consists of a number of program phases each exhibiting their own particular behavior, see for example [74]. As such for a given sample rate, it is more likely that a sampled trace consisting of many small trace samples, such as scenario (ii), will cover all or at least a large part of all the program phases. A sampling regime with fewer large trace samples will probably not cover a large part of the various program phases.

8.1.2 Representative Trace Samples

Obviously, it is very important to select representative samples from a program trace. For example, samples from the beginning of a program trace will only contain non-representative initialization code. This is an example of the more general case that the execution of a computer program consists of a number of program phases, as described by Sherwood et al. [74]. As such, it is very important to select samples in such a way that all program phases are present in the sampled trace proportionally.

In the previous section, we used a periodic or systematic sample selection mechanism. A periodic selection assumes that the samples are chosen at equidistant intervals, i.e., the number of instructions between two consecutive samples is always the same. A possible pitfall is that the inter-sample interval equals the frequency of consecutive program phases or one of its harmonics.

Therefore, some authors have proposed (pseudo-)random sampling by selecting a number of samples randomly from the complete program trace.

Dubey and Nair [19] propose a profile-driven sampling technique that is based on basic block execution profiles. A basic block execution profile measures the number of times each basic block is executed during a program execution. In their approach, Dubey and Nair first measure the basic block execution profile of the original program execution. Subsequently, they scale this basic block execution profile with the simulation speedup they want to attain through trace sampling. For example, if a 10X simulation speedup is pursued, the basic block execution profile is scaled by a factor 10. Subsequently, a sampled trace is generated using this rescaled basic block execution profile, i.e., the occurrence of each basic block in the sampled trace is a factor 10 smaller as it is in the original program execution. This approach has two potential shortcomings. First, using the basic block execution profile as a program characteristic for selecting representative trace samples will yield a sampled trace with a representative basic block profile but does not guarantee representative branch behavior or representative cache behavior. Second, the scaling factor is

chosen arbitrarily based on the simulation time reduction that researchers want to achieve. Although this approach is defendable from a simulation time perspective, is may not be in the perspective of the quality of the sampled trace.

Lauterbach [50] evaluates the representativeness of a sampled trace using the instruction mix, the function execution frequency and cache statistics. His approach works as follows: he starts by taking short trace samples. He subsequently measures the program characteristics mentioned above to evaluate the quality of the sampled trace. If the sampled trace is not representative, samples are added to the sampled trace until the sampled trace is representative.

Iyengar and Trevillyan [42] and Iyengar et al. [43] propose the R-metric to quantify the quality of a sampled trace. The R-metric is based on the notion of fully-qualified basic blocks. A basic block is fully qualified when the branch behavior (taken/not-taken) as well as the cache behavior of this basic block as well as the two or three basic blocks executed before the current is taken into account. The R-metric can be measured on the original trace as well as on the sampled trace. An R-metric for the sampled trace that is close to the R-metric of the original trace designates a representative sampled trace. Iyengar et al. also propose a heuristic algorithm to generate sampled traces based on this R-metric. The main disadvantage of this method however is the huge amount of memory that is required to keep track of all the fully-qualified basic blocks for large applications. The authors report that for gcc they were unable to keep track of all the fully-qualified basic blocks.

Skadron et al. [76] select a single representative sample of 50 million instructions for their microarchitectural simulations. To this end, they first measured branch misprediction rates, data cache miss rates and instruction cache miss rates for each interval of 1 million instructions. By plotting these measurements as a function of the number of instructions simulated they observe the periodic behavior as well as the initialization phase of a program execution. Based on these plots, they manually select a contiguous trace sample of 50 million instructions for each benchmark. Obviously, this trace samples were chosen after the initialization phase. The validation of these 50 million instruction samples was done by comparing the performance characteristics (obtained through detailed microarchitectural simulations) of these sampled traces to 250 million instruction samples. We believe there are several shortcomings and potential weaknesses in this approach. First, the selection of a single contiguous sample of 50 million instructions might not be sufficient to represent the various program phases of a program execution. Second, the size of the sample seems arbitrarily chosen; probably to limit to total simulation time. Third, the selection of the sample was done manually. Fourth, the validation of the sampled trace was done by comparing the sampled trace with another sampled trace. The latter one might not be representative for the complete program execution which might compromise the validation.

Lafage and Seznec [48] use cluster analysis to detect and select similarly behaving trace samples. In their approach, they first measure two microarchitecture-independent metrics for each interval or sample of 1 million instructions. These metrics quantify the temporal and spatial behavior of the data reference stream in each interval. Subsequently, they perform cluster analysis on these data to detect similarly behaving intervals or trace samples. For each cluster, the sample that is closest to the centre of the cluster is chosen as the representative sample for that cluster. During sampled trace simulation, the performance characteristics for each representative sample are then weighted with the number of samples that are contained in the cluster the sample represents. As said above, the metrics that are used by Lafage and Seznec measure microarchitecture-independent locality metrics. Although this approach is appealing, we believe accurate microarchitecture-independent metrics measuring temporal and spatial program behavior are hard to develop. The temporal locality metric that was used by Lafage and Seznec for example is based on the data reuse distance expressed in terms of the number of instructions executed between two accesses to the same address. Consider the following two data reference streams: (i) a b a b a b a b a, and (ii) a b a c a d a e a. Both examples will result in the same locality metric although the first reference stream exhibits significantly more temporal locality than the second one. From this simple example, we conclude that this metric may give a distorted view on the temporal locality behavior of a data reference stream. As such, using this metric for differentiating similarly behaving trace samples may be risky since the two above mentioned examples would be grouped in the same cluster although exhibiting a clearly different locality behavior.

Sherwood et al. [75] also use cluster analysis to detect similarly behaving trace samples. However, the approach taken by Sherwood et al. [75] is different from the approach discussed above. First, they use trace samples containing 100 million instructions instead of 1 million instructions. Second, they evaluate their sampled traces through detailed microarchitecture simulation. Lafage and Seznec on the other hand only considered the data cache. Third, they limit the total size of the sampled trace to 1 billion instructions, i.e., maximum 10 samples of 100 million instructions. Fourth, the program characteristic that is used for discriminating the trace samples is based on basic block execution profiles. Trace samples executing the same basic blocks in the same relative proportion are considered to behave similarly. As argued above, a basic block execution profile might not give an accurate view on other program characteristics that really affect performance, such as branch behavior and data cache behavior. As such, trace samples with a similar basic block execution profile but with different data cache behavior might be grouped together through cluster analysis which might compromise the quality of the sampled trace.

Wunderlich et al. [86] use inferential statistics to ensure the quality of the sampled traces. The quality of the sampled trace is determined by two independent terms,

namely the confidence level $(1 - \alpha)$ and the confidence interval $\pm\varepsilon \cdot \overline{X}$. If the confidence level or the confidence interval are unacceptable for a given sampled trace, the number of samples in the sampled trace has to be increased until both terms meet acceptable levels.

8.1.3 Cold-Start Problem

The fourth major issue that needs to be dealt with for trace sampling, is the correct hardware state at the beginning of each trace sample. This problem is well known in the literature as the *cold-start problem*. The cold-start problem is especially apparent in the case of hardware structures with a large state, such as caches, branch predictors and other kinds of predictors. One possible way of dealing with the cold-start problem is to simulate (without computing performance characteristics) additional references before each sample to warm-up hardware structures. These additional references are part of the *warm-up*. Consequently, simulation using trace samples works as follows, see also Fig. 8. First, a number of instructions are skipped, which we call *cold* simulation. Second, the instructions that are contained in the warm-up are used to update hardware structures but *no* performance characteristics, e.g., cache miss rates and IPC, are being measured. This kind of simulation is called *warm* simulation. The hardware structures being updated are typically structures with a large hardware state, such as cache memories and branch predictors. The third and last step is *hot* simulation in which the instructions are simulated and performance characteristics are computed. The performance metrics obtained from the hot simulations will then be used for reporting performance results. As such, when representative samples are chosen and if the warm-up is highly accurate, these performance results will be very close to the performance results as if we would have simulated the complete original trace.

Several approaches have been proposed to handle the cold-start problem. In this paragraph, these methods are discussed within the context of cache simulation. However, they can be easily generalized for the purpose of full processor simulation.

- The *cold* scheme or the *no warm-up* scheme assumes an empty cache at the beginning of each sample. Obviously, this scheme will overestimate the cache miss rate. However, the bias can be small for large samples.
- Another option is to *checkpoint* or to store the hardware state at the beginning of each sample and impose this state when simulating the sampled trace. This approach yields a perfect warm-up. However, the storage needed to store these checkpoints can explode in case of many samples. In addition, the hardware state of all possible cache configurations needs to be stored. Obviously, the latter constraint implies that the complete trace needs to be simulated for all pos-

sible cache configurations. As such, trace sampling is no longer useful in this scenario.

- A frequently used approach is *stitch* in which the hardware state at the beginning of a sample is approximated with the hardware state at the end of the previous sample. This approach was found to be effective for a suite of Windows NT desktop applications by Crowley and Baer [16]. However, this approach does not guarantee accurate warm-up and can therefore lead to significant errors.
- The *prime-xx%* method assumes an empty cache at the beginning of each sample and uses xx% of each sample to warm-up the cache. Actual simulation then starts after these xx% instructions. The warm-up scheme *prime-50%* is also called *half* in the literature. We can make the following considerations. In some cases the warm-up length can be too short to allow accurate warm-up. In other cases, the warm-up length could be shortened without compromising accuracy.
- A combination of the two previous approaches was proposed by Conte et al. [15]: the state of the cache at the beginning of each sample is the state at the end of the previous sample plus warming-up using a fraction of the sample.
- Another approach proposed by Kessler et al. [45] and Wood et al. [85] is to assume an empty cache at the beginning of each sample and to estimate which cold-start misses would have been avoided if the cache state at the beginning of the sample was known.
- Nguyen et al. [61] use W instructions to warm-up the cache which is calculated as follows:

$$W = \frac{C/L}{m \cdot r},$$

with C the cache size, L the line size, m the cache miss rate and r the memory reference ratio, or in other words the percentage loads and stores in case of a data cache evaluation. The problem with this approach is that the cache miss rate m is unknown; this is exactly what we are trying to approximate through trace sampling. A possible solution is to use the cache miss rate under the *no-warmup* scheme. However, this might lead to a warm-up length that is too short because the miss rate under the no-warmup scenario is generally an overestimation.

- Minimal Subset Evaluation (MSE) proposed by Haskins Jr. and Skadron [33] determines a minimally sufficient warm-up period. The technique works as follows. First, the user specifies the probability that the cache state at the beginning of the sample after warm-up equals the cache state at the beginning of the sample in case of full trace simulation and thus perfect warm-up. Second, the MSE formulas are used to determine how many unique references are required dur-

ing warm-up. Third, using a benchmark profile it is calculated where exactly the warm-up should be started in order to generate these unique references.

- Haskins Jr. and Skadron [34] continued this research project and proposed Memory Reference Reuse Latency (MRRL) as a refinement of MSE, i.e., by reducing the total warm-up length without sacrificing accuracy. In this approach, a histogram of the MRRL, or the number of memory references between two references to the same memory location, is used to determine when to start the warm simulation.
- Very recently, Eeckhout et al. [23] proposed yet another warm-up technique that is similar to the two previous techniques, MSE and MRRL, in that a reliable indication should be given beforehand whether the hardware state will be accurately approximated at the beginning of a sample. However, the mechanism to achieve this is quite different from the ones employed in MSE and MRRL. Whereas MSE looks at the references in the pre-sample, and MRRL looks at the references in the pre-sample and the sample, the approach presented by Eeckhout et al. [23] is based on the references in the sample solely. This approach will be discussed in the following paragraph.

Example of a Warmup Strategy

The rationale behind the approach proposed by Eeckhout et al. [23] is that the cache memories need only to be warmed-up with memory addresses that are referenced within the sample. For example, consider the case where the working set—which is defined as the set of unique references—of a sample is very small. In other words, the number of sets in the cache that will be touched by this working set is limited. As such, the other sets in the cache need not to be warmed-up since this will not have any impact on the accuracy of the sample. In addition, keeping this warm-up as small as possible helps in shortening the total warm-up length and thus the total simulation time.

The mechanism of our new warm-up technique is explained in Fig. 12: a sample is presented as well as its pre-sample. Consider the unique memory references contained in the sample, e.g., a, b, c, d, e and g in Fig. 12. For each unique reference in the sample, consider the last reference to the same memory address in the pre-sample. This is shown through the arrows in Fig. 12. For example, there are two references to a in the sample and four references to a in the pre-sample; the arrow points to the fourth a in the pre-sample. Consequently, a perfect warm-up is obtained for the sample by starting the warm simulation phase at the first memory reference in the pre-sample having an arrow pointing to it. Indeed, all the unique references in the sample will be touched in the warm-up. In addition, all the memory references

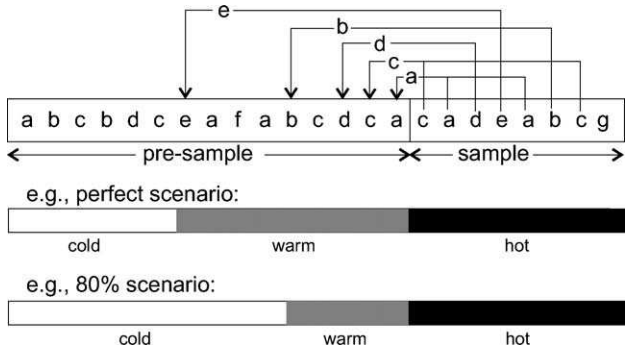


FIG. 12. Warm-up based on memory reference reuse latencies of unique references within a sample.

potentially conflicting with these warm-up references, e.g., reference f in the pre-sample in Fig. 12, are included in the warm-up as well. Note also that a reference in the sample that remains untouched in the pre-sample, e.g., reference g in Fig. 12, does not affect the warm-up length nor the accuracy.

From experiments as described in [23], a highly accurate warm-up can be obtained without considering *all* the unique references in the sample. This can potentially lead to a higher simulation time reduction without sacrificing too much accuracy. For example, we could consider only 80% of these unique references. More precisely, we select 80% of the references in the pre-sample that have an arrow pointing to them and that are closest to the beginning of the sample. E.g., in Fig. 12, the 80% scenario starts the warm simulation phase at the last reference to memory address b yielding a warm-up of 5 instructions; the perfect (100%) scenario starts the warm simulation phase at the last reference to memory address e yielding a warm-up of nine instructions.

Evaluation

In Fig. 13, the data cache miss rate is shown as a function of the cache size and as a function of the warm-up method. These graphs show three example SPEC CPU2000 integer benchmarks, namely vpr, parser and bzip2. The various warm-up methods that are considered in these graphs are:

- *full warm-up*: this means that all the instructions in the pre-sample are warm instructions. As such, the hardware state of the cache at the beginning of the sample is perfectly warmed up.
- *no warm-up*: all the instructions in the pre-sample are cold instructions. In addition, we empty the cache at the beginning of the sample. As can be seen from Fig. 13, this approach can lead to severe inaccuracies.

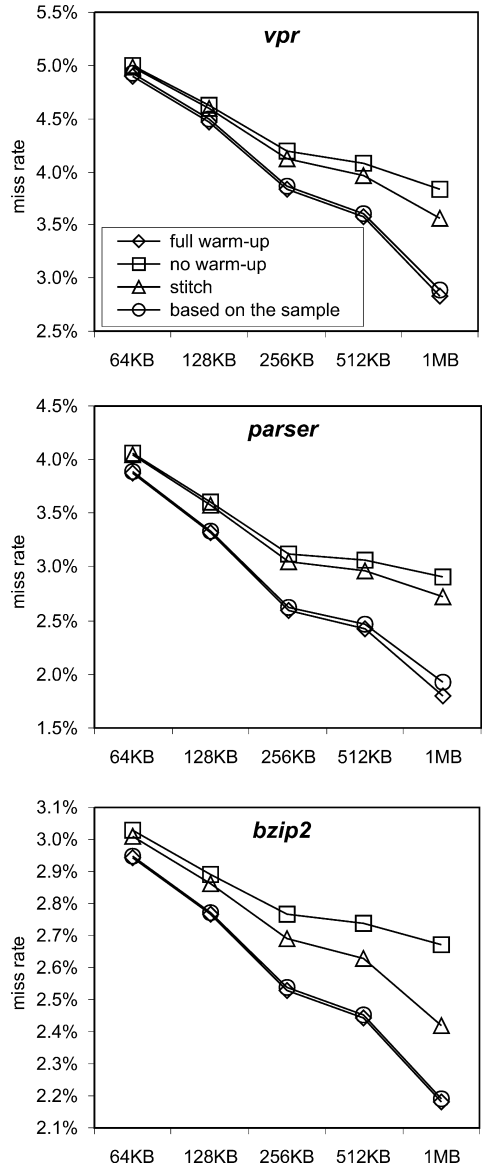


FIG. 13. Dealing with the cold-start problem: (i) full warm-up (no cold instructions), (ii) no warm-up (no warm instructions and empty cache at beginning of each sample), (iii) stitch (no warm instructions and stale cache state at beginning of each sample), and (iv) the warm-up method proposed by Eeckhout et al. [23] that is based on the references in the sample.

- *stitch*: all the instructions in the pre-sample are cold instructions. However, the hardware state of the cache at the beginning of the sample is chosen to be the same as at the end of the previous sample. This warm-up method is also inaccurate for the examples mentioned in Fig. 13. However, for some benchmarks this approach can be highly accurate.
- *based on the sample*: this warm-up method is based on the memory references that occur in the sample itself as discussed above. For this curves, 90% of the unique references in pre-sample are considered. Fig. 13 clearly shows that this warm-up method attains high accuracies. For a more elaborate evaluation of this warm-up method, we refer to [23].

The *simulation time speedup* as a result of trace sampling is the ratio of the total number of references in the original trace versus the number of warm and hot simulation references. In other words, if we are able to consider 4% of the instructions in the original trace as warm or hot simulation instructions, then the simulation time speedup is a factor 25. Generally, a higher sample rate—the hot simulation parts versus the complete trace—leads to a higher accuracy at the cost of a longer simulation time. The same can be said about the warm-up length: a larger warm-up length generally leads to a higher accuracy at the cost of a longer simulation time. This is illustrated in Fig. 14 where the miss rate is shown under three warmup scenarios: (i) no warmup, (ii) full warmup and (iii) the warmup mechanism as presented by Eeckhout et al. [23]. This is done for ten SPEC CPU2000 integer benchmarks and for a 1 MB eight-way set-associative data cache with a block size of 32 bytes. The Y axes represent the cache miss rate whereas the X axes represent the simulation time speedup. The solid lines in these graphs represent the cache miss rates that are obtained under the full warmup scenario. The dashed lines represent the cache miss rates under the no warmup scenario. The simulation time speedup that can be obtained using this approach is a factor 500 for our experimental setup. The various points for the 'warmup' curves correspond varying percentages of unique references considered in the pre-sample, i.e., 50%, 60%, 70%, 80%, 90%, 95% and 99%. Note that the data are heavily dependent on the chosen benchmark. For example, for bzip2 a simulation time speedup of a factor 9 can be attained while preserving a high prediction accuracy. For twolf on the other hand, the simulation time speedup that can be attained is much larger, up to a factor 200.

8.2 Reduced Input Sets

Recall that this section deals with reducing the length of a program run. Next to trace sampling, there exists another approach with the same goal, namely *reduced input sets*. KleinOsowski and Lilja [47] propose to reduce the simulation time of the

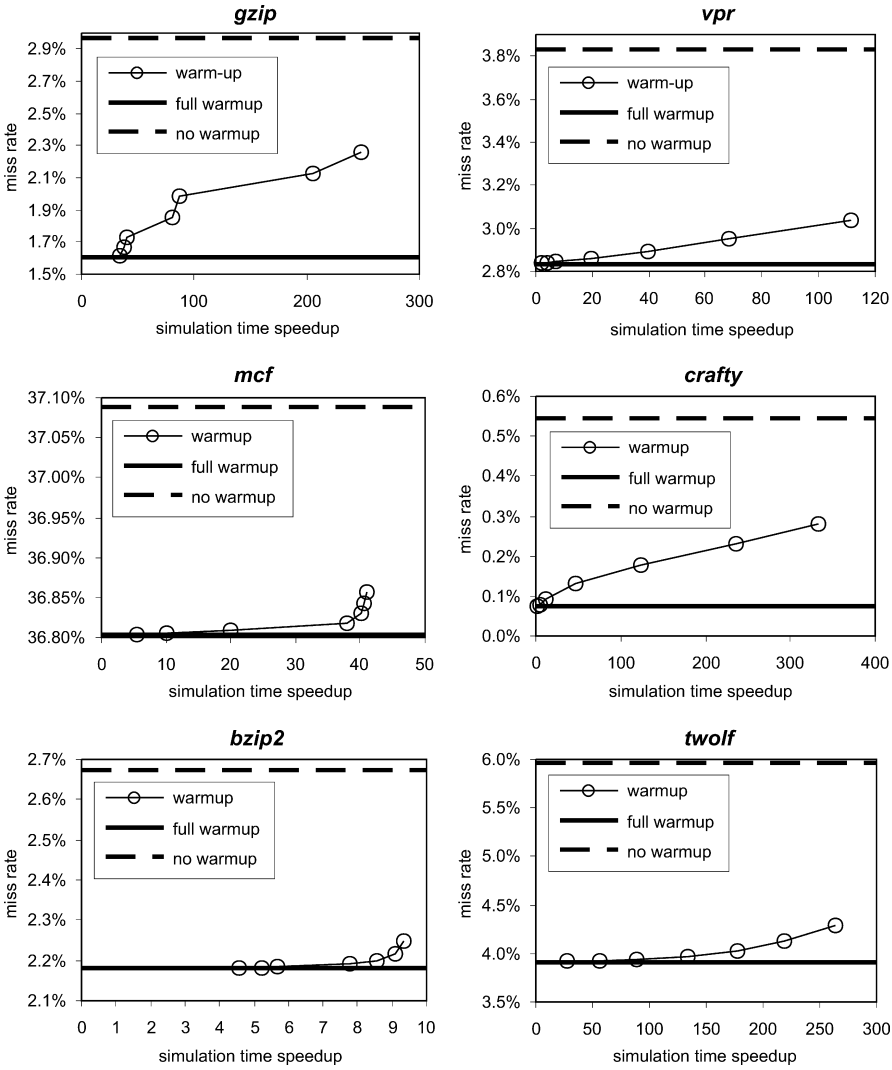


FIG. 14. Simulation time speedup versus accuracy for the SPEC CPU2000 integer benchmarks and a 1 MB eight-way set-associative data cache under three warmup scenarios: (i) no warmup, (ii) full warmup and (iii) warmup as presented by Eeckhout et al. [23].

SPEC 2000 benchmark suite[5] by using reduced input data sets, called MinneSPEC.[6]

[5] http://www.spec.org.

[6] http://www-mount.ee.umn.edu/~lilja/spec2000/.

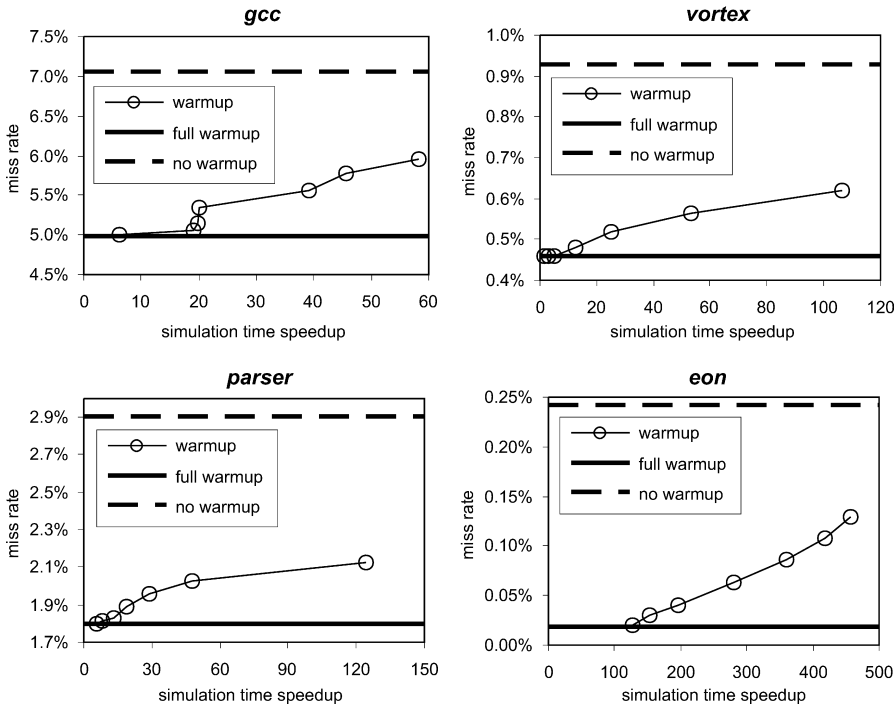


FIG. 14. — *(Continued)*

These reduced input sets are derived from the reference inputs by a number of techniques: modifying inputs (for example, reducing the number of iterations), truncating inputs, etc. The benefit of these reduced inputs is that the dynamic instruction count when simulating these inputs can be significantly smaller than the reference inputs. They propose three reduced inputs: smred for short simulations (100 million instructions), mdred for medium length simulations (500 million instructions) and lgred for full length, reportable simulations (1 billion instructions). For determining whether two input sets result in more or less the same behavior, they used the chi-squared statistic based on the function-level execution profiles for each input set. A function-level profile is nothing more than a distribution that measures what fraction of the time is spent in each function during the execution of the program. Measuring these function-level execution profiles was done using the UNIX utility `gprof`. Note however that a resemblance of function-level execution profiles does not necessarily imply a resemblance of other workload characteristics that are probably more closely related to performance, such as instruction mix, cache behavior,

branch predictability, etc. For example, consider the case that we would scale down the number of times a function is executed by a factor S. Obviously, this will result in the same function-level execution profile. However, a similar function-level execution profile does not guarantee a similar behavior concerning for example the data memory. Indeed, reducing the input set often reduces the size of the data the program is working on while leaving the function-level execution profile untouched. And this might lead to a significantly different data cache behavior. KleinOsowski and Lilja [47] also recognized that this is a potential problem.

Eeckhout et al. [24] validated the program behavior similarity of MinneSPEC versus the reference SPEC benchmark suite using the methodology that is based on PCA and CA. Recall that this methodology transforms a highly correlated, high dimensional workload space into a non-correlated, low dimensional workload space. In this understandable workload space, program-input pairs that are close to each other exhibit similar behavior; program-input pairs far away from each other reflect dissimilar behavior. As such, this methodology can be used to validate reduced input sets. Indeed, if a reduced input is situated close to its reference counterpart we conclude that the reduced input results in similar behavior. The results from the analysis of MinneSPEC are shown in Table III. In these tables, we verify whether the reduced input sets as proposed by MinneSPEC resemble their reference counterparts. For each SPEC CPU2000 benchmark for which MinneSPEC provides reduced inputs, we have compared the program behavior resulting from these reduced inputs to the reference inputs. More specifically, we identify three possible classifications: (i) similar behavior (label 'S'), (ii) more or less similar behavior (label 'M') and (iii) dissimilar behavior (label 'D'). For example, see Table III, although the lgred input for parser has a dynamic instruction count that is at least a factor 100 smaller than the reference input and a data memory footprint that is nearly a factor 2 smaller, lgred results in a similar program behavior. As such, using the lgred input instead of the reference input will result a simulation speedup of a factor 100 without losing accuracy. The smred input for parser on the other hand, yields a program behavior that is dissimilar to the reference input and should therefore not be considered as a viable alternative for the reference input.

From Table III, we can make some general conclusions. First, the lgred input is generally the best input among the inputs proposed in MinneSPEC. In other words, lgred generally yields (more or less) similar behavior. Unfortunately, there are a few exceptions, namely mcf, gcc and vortex for which lgred yields dissimilar behavior. Second, the smallest inputs smred and mdred generally lead to a dissimilar behavior, except for gzip.source, gzip.random and gzip.log.

If we compare the results by KleinOsowski and Lilja [47] with the results in Table III, we observe that in most cases the results agree very well. However, in a num-

TABLE III
CHARACTERISTICS OF THE BENCHMARKS USED WITH THEIR INPUTS, DYNAMIC INSTRUCTION COUNT (IN MILLIONS), THE DATA MEMORY FOOTPRINT IN 64-BIT WORDS (IN THOUSANDS) AND WHETHER THE REDUCED INPUTS ARE SIMILAR TO THE REFERENCE INPUT ACCORDING TO OUR ANALYSIS: 'S' STANDS FOR SIMILAR, 'M' STANDS FOR MORE OR LESS SIMILAR, AND 'D' STANDS FOR DISSIMILAR

Benchmark	Input	dyn(M)	mem(K)	Similar?
vpr.route	ref	94,331	6,300	
	train	10,457	1,277	S
	lgred (test)	857	229	S
	mdred	92	81	D
	smred	6	20	D
vpr.place	ref	112,264	376	
	train	14,009	89	S
	lgred (test)	1,566	27	D
	mdred	224	16	D
	smred	18	12	D
twolf	ref	346,485	642	
	train	13,200	356	S
	lgred	973	51	M
	mdred (test)	259	20	D
	smred	92	15	D
parser	ref	546,748	4,770	
	train	13,433	3,313	S
	test	4,203	1,290	M
	lgred	4,527	2,814	S
	mdred	612	1,289	M
	smred	269	738	D
mcf	ref	61,867	24,883	
	train	9,168	24,397	D
	lgred	794	24,205	D
	mdred (test)	260	24,151	D
	smred	189	24,144	D
gcc	ref.scilab	62,031	14,774	
	ref.integrate	13,164	9,328	
	ref.expr	12,086	6,837	
	ref.200	108,670	14,254	
	ref.166	46,918	20,218	
	test.cccp	2,016	854	D
	lgred.cp-decl (train)	5,117	2,185	D
	mdred.rtlanal	551	385	D
	smred.c-iterate	97	260	D

(*continued on next page*)

TABLE III — *Continued from previous page*

Benchmark	Input	dyn(M)	mem(K)	Similar?
bzip2	train	61,128	3,343	S
	test	8,822	1,920	D
bzip2.source	ref	108,878	10,689	
	lgred	1,820	1,546	S
bzip2.program	ref	124,927	12,288	
	lgred	2,159	1,575	S
bzip2.graphic	ref	143,565	14,608	
	lgred	2,644	1,626	S
vortex	ref1	118,977	9,382	
	ref2	138,678	7,869	
	ref3	133,044	9,298	
	train	17,813	2,734	S
	test	9,808	2,893	M
	lgred	1,154	1,604	D
	mdred	415	1,099	M
	smred	88	1,025	D
perlbmk	ref.splitmail.957	110,829	13,848	
	ref.splitmail.850	127,498	17,357	
	ref.splitmail.704	66,753	9,105	
	ref.splitmail.535	63,693	9,244	
	ref.perfect	29,064	56	
	train.scrabbl	27,648	60	D
	train.perfect	17,218	55	D
	train.diffmail	35,630	8,748	D
	test	5	39	S
	lgred (ref.makerand)	2,009	1,058	M
	mdred	929	1,058	M
	smred	201	1,057	M
gzip.source	ref	84,367	10,521	
	lgred	1,583	234	S
	mdred	1,552	219	S
	smred	1,486	220	S
gzip.random	ref	82,167	15,834	
	lgred	1,361	297	S
	mdred	1,362	297	S
	smred	1,362	297	S
gzip.program	ref	168,868	11,703	
	lgred	2,858	234	S
	mdred	2,732	209	M
	smred	4,025	217	M

(*continued on next page*)

TABLE III — *Continued from previous page*

Benchmark	Input	dyn(M)	mem(K)	Similar?
gzip.log	ref	39,527	9,071	
	lgred	593	185	M
	mdred	597	185	M
	smred	602	188	S
gzip.graphic	ref	103,706	15,230	
	lgred	1,786	272	S
	mdred	1,209	223	S
	smred	4,984	231	D
	train.combined	57,970	7,578	S
	test.combined	3,367	560	S

ber of cases there is a difference between the MinneSPEC results and our results, e.g.,

- the lgred input of vpr.place was found to be similar to the ref input by KleinOsowski and Lilja [47]; our results on the other hand show a dissimilar behavior.
- the same is true for vortex.lgred and gcc.lgred.
- the reverse also happens, where the MinneSPEC results suggest a dissimilar behavior whereas our results reveal a similar behavior: for example, the train input of bzip2 and the reduced inputs for perlbmk.

As such, we can conclude that comparing input sets based on function profiles is quite accurate in general. However, in a number of cases similar function profiles may still result in dissimilar program behavior and vice versa which warns us to be careful when comparing input sets using function profiles only.

It is also interesting to point to the following particularities that we observed through our analysis:

- The reference inputs of vortex result in very similar behavior which raises the question whether it is useful to simulate all the reference inputs during architectural simulations.
- For vpr on the other hand, the place and route reference inputs result in dissimilar behavior. So, we advise researchers to consider both in architectural simulations.
- Note that for vortex the mdred is closer to the ref input than the lgred input, although having a smaller dynamic instruction count. As such, mdred is a better candidate for architectural simulations than lgred.

- The reduced inputs for perlbmk are very similar to each other. However, they are quite different from the reference inputs. The test input on the other hand, resembles quite well the ref.perfect input.

From this section, we conclude that for most benchmarks it is possible to propose reduced input sets that result in a much smaller dynamic instruction count but yield a similar program behavior as the reference input. Evaluating the (dis)similarity between reduced inputs and reference inputs should be done using characteristics that reflect real program behavior instead of function-level execution profiles. The statistical analysis methodology that is based on PCA and CA can be successfully used for this purpose.

8.3 Comparing Trace Sampling Versus Reduced Input Sets

The previous two sections discussed two techniques to reduce the length of a program simulation run, namely trace sampling and reduced input sets. From the results presented in these sections, we concluded that both techniques can significantly reduce the total simulation time and can yield representative program behavior. However, it is unclear which method is to be preferred. In other words, which method yields a program behavior that most closely resembles the reference input while yielding a significant simulation speedup.

To compare trace sampling versus reduced input sets, we can make use of the methodology presented in Section 7. Using this approach for the purpose of this section is thus straightforward. If a sampled trace is closer to the reference input than a reduced input in the design space obtained from the analysis, we can conclude that trace sampling results in more representative program behavior than the reduced input, and vice versa. For this section, we used a number of long running SPEC CPU95 benchmarks (m88ksim, vortex, go and compress) as well as a database postgres running a decision support TPC-D query, namely query 16. For each of these benchmarks we have considered three sampled traces: one with a sample rate of 10%, a second one with a sample rate of 1% and a third one with a sample rate of 0.1%. These sampled traces were obtained using the reference inputs. We assumed a periodic sampling regime and samples of 1 million instructions each. Next to these sampled traces, we also considered a number of reduced input sets. For m88ksim, vortex and go, we considered the train input. For compress, we generated a number of reduced inputs by modifying the reference input.

The results from these analyses can be represented in a dendrogram which represents the (dis)similarity between points in the workload design space. From Fig. 15, we can make the following conclusions for the various benchmarks:

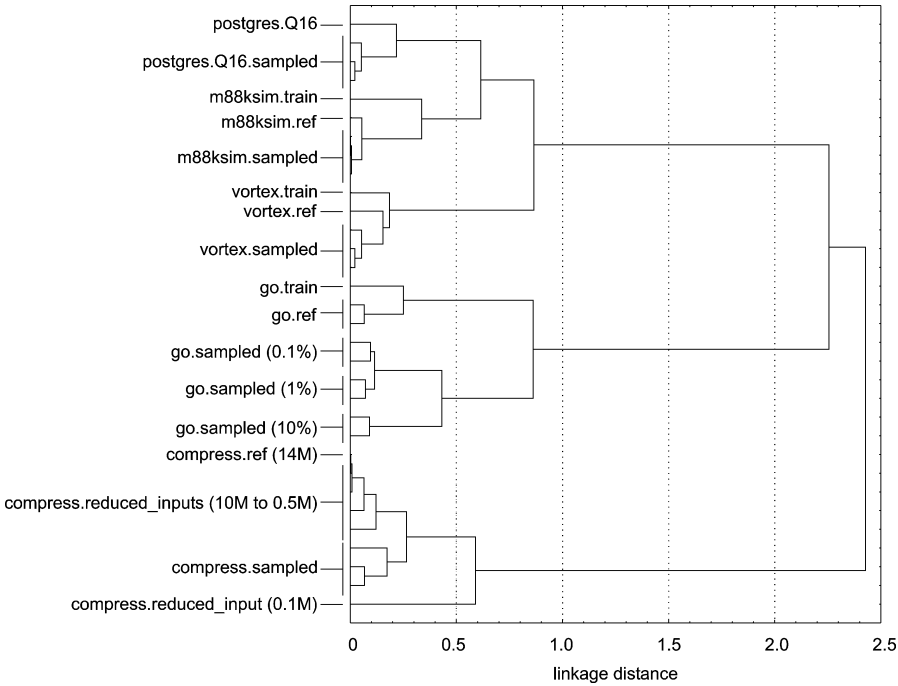


FIG. 15. Comparing trace sampling versus reduced inputs for a number of SPEC CPU95 integer benchmarks and a database postgres running TPC-D query Q16.

- For some benchmarks, trace sampling results in program behavior that is more similar to the reference input than with reduced inputs. Indeed, for m88ksim and vortex the linkage distances between the points corresponding to the sampled traces and the reference input are much shorter than between the reduced inputs and the reference input. As such, for these benchmarks, trace sampling is definitely a better option than reduced inputs.
- For other benchmarks on the other hand, for example go and compress, reduced inputs result in program behavior that is more similar to the reference input than what can be attained through trace sampling. For compress, we used a variety of reduced inputs. Note however, that not all reduced inputs result in program behavior that is similar to the reference input. The smallest reduced input 100,000 e 2231 (which is derived from the reference input 14,000,000 e 2231) results in very dissimilar behavior. The other reduced input sets, varying from 500,000 e 2231 to 10,000,000 e 2231, yield similar behavior. As such, we conclude that the reduced input should not be smaller than 500,000 e 2231.

TABLE IV
COMPARING TRACE SAMPLING VERSUS REDUCED INPUTS: THE METHOD OF CHOICE (SHOWN IN BOLD) DEPENDS ON THE BENCHMARK

Benchmark	Input/sampling regime	dyn(M)	Speedup
postgres	TPC-D query 16	82,228	
	10%-sampled trace	8,222	10
	1%-sampled trace	822	100
	0.1%-sampled trace	**82**	**1,000**
m88ksim	ref	71,161	
	train	24,959	2.9
	10%-sampled trace	7,116	10
	1%-sampled trace	711	100
	0.1%-sampled trace	**71**	**1,000**
vortex	ref	92,555	
	train	3,244	28.5
	10%-sampled trace	9,255	10
	1%-sampled trace	925	100
	0.1%-sampled trace	**92**	**1,000**
go	ref1 (50 21 9stone21.in)	35,758	
	10%-sampled ref1 trace	3,575	10
	1%-sampled ref1 trace	357	100
	0.1%-sampled ref1 trace	35	1,000
	ref2 (50 21 5srone21.in)	35,329	
	10%-sampled ref2 trace	3,532	10
	1%-sampled ref2 trace	353	100
	0.1%-sampled ref2 trace	35	1,000
	train	**593**	**60.3**
compress	ref (14,000,000 e 2231)	60,102	
	10%-sampled trace	6,010	10
	1%-sampled trace	601	100
	0.1%-sampled trace	60	1,000
	10,000,000 e 2231	42,936	1.4
	5,000,000 e 2231	21,495	2.8
	1,000,000 e 2231	4,342	13.8
	500,000 e 2231	**2,182**	**27.5**
	100,000 e 2231	423	142

Table IV summarizes the conclusions from this analysis. The method of choice, trace sampling versus reduced input sets, is shown in bold for each benchmark. In addition, the simulation time speedup that is obtained for each method is displayed as well.

As a general conclusion from this section, we can state that for some benchmarks trace sampling is more accurate than reduced input sets. For other benchmarks, the

opposite is true. As such, we recommend researchers working with either sampled traces or reduced input sets to verify whether they are using an approach that yields representative behavior. The methodology presented here is an excellent tool for this purpose.

Haskins Jr. et al. [36] also present a comparison between trace sampling and reduced input sets. They take the same conclusions, namely that both methods can result in the most accurate performance predictions. In other words, for some benchmarks trace sampling is the method of choice whereas for other benchmarks reduced inputs should be employed. In addition, they argue that both methods come with their own benefits. Reduced inputs allow the execution of a program from the beginning to the end. Trace sampling allows flexibility by varying the sample rate, the sample length, the number of samples, etc.

9. Increasing the Simulation Speed

The fourth and last aspect that contributes to the time-consuming behavior of architectural simulations is the slowdown of the simulator itself. As stated in Section 4.4, the slowdown factor of an architectural simulator is typically 50,000 to 300,000. Obviously, there is only one solution to this problem, namely to optimize the simulator. However, there are two possible constraints: (i) without sacrificing accuracy, and (ii) with sacrificing (little) accuracy.

The first approach we discuss does not sacrifice accuracy. Schnarr and Larus [73] show how to speed up an architectural simulator using memoization. Traditionally, memoization refers to caching function return values in functional programming languages. These cached values can then be returned when available during execution avoiding expensive computations. Schnarr and Larus [73] present a similar technique that caches microarchitecture states and the resulting simulation actions. When a cached state is encountered during simulation, the simulation is then fast-forward by replaying the associated simulation actions at high speed until a previously unseen state is reached. They achieve an 8 to 15 times speedup over SimpleScalar's out-of-order simulator by Burger and Austin [11], while producing exactly the same result.

The following three approaches sacrifice little accuracy, i.e., these approaches model a microarchitecture at a slightly higher abstraction level which obviously introduce additional modeling errors. Bose [6] proposes to *pre-process* a program trace, e.g., by tagging loads and stores with hit/miss information, or by tagging branches with prediction information (wrong or correct prediction). This tagged program trace can then be executed on a simulator that imposes an appropriate penalty

in the simulator just as is the case with statistical simulation, Section 6.1. The simulator speedup comes from the fact that several hardware structures do not need to be modeled.

Loh [51] presents a time-stamping algorithm that achieves an average prediction error of 7.5% with a 2.42X simulation speedup for wide-issue superscalar architectures. This approach is built on the idea that it is sufficient to know *when* events—such as the end of an instruction execution or the availability of a resource—occur. By time-stamping the resources associated with these events, the IPC can be computed by dividing the number of instructions simulated by the highest time stamp. The inaccuracy comes from making assumptions in the time-stamping algorithm which make it impossible to accurately model the behavior of a complex out-of-order architecture such as out-of-order cache accesses, wrong path cache accesses, etc.

Ofelt and Hennessy [67] present a profile-based performance prediction technique that is an extension of a well known approach consisting of two phases. The first (instrumentation) phase counts the number of times a basic block is executed. The second phase then simulates each basic block while measuring the amount of IPC. This estimated IPC number is multiplied by the number of times the basic block is executed during a real program execution. The sum over all basic blocks then gives an estimate of the IPC of program. The approach presented by Ofelt and Hennessy [67] extends this simple method to enable the modeling of out-of-order architectures, e.g., by modeling parallelism between instructions from various basic blocks. This approach achieves a high accuracy (errors of only a few percent are reported) while assuming perfect branch prediction and perfect caches. However, when a realistic branch predictor and realistic caches are included in the evaluation, the accuracy falls short as described by Ofelt [66].

10. Conclusion

The architectural simulations that need to be done during the design of a microprocessor are extremely time-consuming for a number of reasons. First, the microarchitectural design space that needs to be explored is huge. Second, the workload space or the number of benchmarks (with suitable inputs) is large as well. Third, the number of instructions that need to be simulated per benchmark is huge. Fourth, due to the ever increasing complexity of current high performance microprocessors, simulators are running relatively slower (more simulator instructions per simulated instruction). As such, computer architects need techniques that could speed up their design flow. In this chapter, we presented such an architectural design framework. First, a region of interest is identified in the huge design space. This is done through

statistical simulation which is a fast and quite accurate simulation technique. Subsequently, this region is further evaluated using detailed architectural simulations. Fortunately, the duration of these slow simulation runs can be reduced: (i) by selecting a limited but representative workload using statistical data analysis techniques, such as principal components analysis and cluster analysis, (ii) by limiting the number of instructions that need to be simulated per benchmark in the workload; this can be done through trace sampling or through the use of reduced input sets, and (iii) by optimizing the architectural simulations so that the simulator's instruction throughput increases. As such, we conclude that this architectural simulation methodology can reduce the total design time of a microprocessor considerably.

REFERENCES

[1] Austin T., Larson E., Ernst D., "SimpleScalar: An infrastructure for computer system modeling", *IEEE Computer* **35** (2) (Feb. 2002) 59–67.

[2] Bechem C., Combs J., Utamaphetai N., Black B., Blanton R.D.S., Shen J.P., "An integrated functional performance simulator", *IEEE Micro* **19** (3) (May/June 1999) 26–35.

[3] Black B., Shen J.P., "Calibration of microprocessor performance models", *IEEE Computer* **31** (5) (May 1998) 59–65.

[4] Bose P., "Performance test case generation for microprocessors", in: *Proceedings of the 16th IEEE VLSI Test Symposium*, Apr. 1998, pp. 50–58.

[5] Bose P., "Performance evaluation and validation of microprocessors", in: *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'99)*, May 1999, pp. 226–227.

[6] Bose P., "Performance evaluation of processor operation using trace pre-processing", US Patent 6,059,835, May 2000.

[7] Bose P., Conte T.M., "Performance analysis and its impact on design", *IEEE Computer* **31** (5) (May 1998) 41–49.

[8] Bose P., Conte T.M., Austin T.M., "Challenges in processor modeling and validation", *IEEE Micro* **19** (3) (May 1999) 9–14.

[9] Brooks D., Bose P., Schuster S.E., Jacobson H., Kudva P.N., Buyuktosunoglu A., Wellman J.-D., Zyuban V., Gupta M., Cook P.W., "Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors", *IEEE Micro* **20** (6) (November/December 2000) 26–44.

[10] Brooks D., Martonosi M., Bose P., "Abstraction via separable components: An empirical study of absolute and relative accuracy in processor performance modeling", Tech. Rep. RC 21909, IBM Research Division, T.J. Watson Research Center, Dec. 2000.

[11] Burger D.C., Austin T.M., "The SimpleScalar tool set", *Computer Architecture News* (1997). For more information see also http://www.simplescalar.com.

[12] Carl R., Smith J.E., "Modeling superscalar processors via statistical simulation", in: *Workshop on Performance Analysis and its Impact on Design (PAID-98), held in con-*

junction with the 25th Annual International Symposium on Computer Architecture (ISCA-25), Jun. 1998.

[13] Conte T.M., "Systematic computer architecture prototyping", Ph.D. thesis, University of Illinois at Urbana–Champaign, 1992.

[14] Conte T.M., Hirsch M.A., Hwu W.W., "Combining trace sampling with single pass methods for efficient cache simulation", *IEEE Trans. Comput.* **47** (Jun. 1998) 714–720.

[15] Conte T.M., Hirsch M.A., Menezes K.N., "Reducing state loss for effective trace sampling of superscalar processors", in: *Proceedings of the 1996 International Conference on Computer Design (ICCD-96)*, Oct. 1996, pp. 468–477.

[16] Crowley P., Baer J.-L., "Trace sampling for desktop applications on Windows NT", in: *Proceedings of the First Workshop on Workload Characterization (WWC-1998) held in conjunction with the 31st ACM/IEEE Annual International Symposium on Microarchitecture (MICRO-31)*, Nov. 1998.

[17] Desikan R., Burger D., Keckler S.W., "Measuring experimental error in microprocessor simulation", in: *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA-28)*, Jul. 2001, pp. 266–277.

[18] Dubey P.K., Adams III G.B., Flynn M.J., "Instruction window size tradeoffs and characterization of program parallelism", *IEEE Trans. Comput.* **43** (4) (Apr. 1994) 431–442.

[19] Dubey P.K., Nair R., "Profile-driven sampled trace generation", Tech. Rep. RC 20041, IBM Research Division, T.J. Watson Research Center, Apr. 1995.

[20] Edler J., Hill M.D., "Dinero IV trace-driven uniprocessor cache simulator", Available through http://www.cs.wisc.edu/~markhill/DineroIV, 1998.

[21] Eeckhout L., "Accurate statistical workload modeling", Ph.D. thesis, Ghent University, Belgium, available at http://www.elis.rug.ac.be/~leeckhou, Dec. 2002.

[22] Eeckhout L., De Bosschere K., "Hybrid analytical-statistical modeling for efficiently exploring architecture and workload design spaces", in: *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT-2001)*, Sep. 2001, pp. 25–34.

[23] Eeckhout L., Eyerman S., Callens B., De Bosschere K., "Accurately warmed-up trace samples for the evaluation of cache memories", in: *Proceedings of the 2003 High Performance Computing Symposium (HPC-2003)*, Apr. 2003, pp. 267–274.

[24] Eeckhout L., Vandierendonck H., Bosschere K.D., "Designing workloads for computer architecture research", *IEEE Computer* **36** (2) (Feb. 2003) 65–71.

[25] Eeckhout L., Vandierendonck H., Bosschere K.D., "Quantifying the impact of input data sets on program behavior and its applications", *Journal of Instruction-Level Parallelism* **5** (Feb. 2003), http://www.jilp.org/vol5.

[26] Eeckhout L., Vandierendonck H., De Bosschere K., "Workload design: Selecting representative program-input pairs", in: *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques (PACT-2002)*, Sep. 2002, pp. 83–94.

[27] Emer J., Ahuja P., Borch E., Klauser A., Luk C.-K., Manne S., Mukherjee S.S., Patil H., Wallace S., Binkert N., Espasa R., Juan T., "Asim: A performance model framework", *IEEE Computer* **35** (2) (Feb. 2002) 68–76.

[28] Emma P.G., "Understanding some simple processor-performance limits", *IBM Journal of Research and Development* **41** (3) (May 1997) 215–232.

[29] Flynn M.J., Hung P., Rudd K.W., "Deep-submicron microprocessor design issues", *IEEE Micro* **19** (4) (July/August 1999) 11–22.
[30] Franklin M., Sohi G.S., "ARB: A hardware mechanism for dynamic reordering of memory references", *IEEE Trans. Comput.* **45** (5) (May 1996) 552–571.
[31] Gibson J., Kunz R., Ofelt D., Horowitz M., Hennessy J., Heinrich M., "FLASH vs. (simulated) FLASH: Closing the simulation loop", in: *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, Nov. 2000, pp. 49–58.
[32] Hartstein A., Puzak T.R., "The optimal pipeline depth for a microprocessor", in: *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA-29)*, May 2002, pp. 7–13.
[33] Haskins Jr. J.W., Skadron K., "Minimal subset evaluation: Rapid warm-up for simulated hardware state", in: *Proceedings of the 2001 International Conference on Computer Design (ICCD-2001)*, Sep. 2001, pp. 32–39.
[34] Haskins Jr. J.W., Skadron K., "Memory reference reuse latency: Accelerated sampled microarchitecture simulation", Tech. Rep. CS-2002-19, Department of Computer Science, University of Virginia, Jul. 2002.
[35] Haskins Jr. J.W., Skadron K., "Memory reference reuse latency: Accelerated warmup for sampled microarchitecture simulation", in: *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2003)*, Mar. 2003, pp. 195–203.
[36] Haskins Jr. J.W., Skadron K., KleinOsowski A.J., Lilja D.J., "Techniques for accurate, accelerated processor simulation: An analysis of reduced inputs and sampling", Tech. Rep. CS-2002-01, University of Virginia—Dept. of Computer Science, Jan. 2002.
[37] Hennessy J.L., Patterson D.A., *Computer Architecture: A Quantitative Approach*, third ed., Morgan Kaufmann, San Mateo, CA, 2003.
[38] Henning J.L., "SPEC CPU2000: Measuring CPU performance in the new millennium", *IEEE Computer* **33** (7) (Jul. 2000) 28–35.
[39] Hinton G., Sager D., Upton M., Boggs D., Carmean D., Kyker A., Roussel P., "The microarchitecture of the Pentium 4 processor", *Intel Technology Journal* **Q1** (2001).
[40] Hsieh C., Pedram M., "Micro-processor power estimation using profile-driven program synthesis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **17** (11) (Nov. 1998) 1080–1089.
[41] Hughes C.J., Pai V.S., Ranganathan P., Adve S.V., "Rsim: Simulating shared-memory multiprocessors with ILP processors", *IEEE Computer* **35** (2) (Feb. 2002) 40–49.
[42] Iyengar V.S., Trevillyan L.H., "Evaluation and generation of reduced traces for benchmarks", Tech. Rep. RC 20610, IBM Research Division, T.J. Watson Research Center, Oct. 1996.
[43] Iyengar V.S., Trevillyan L.H., Bose P., "Representative traces for processor models with infinite cache", in: *Proceedings of the Second International Symposium on High-Performance Computer Architecture (HPCA-2)*, Feb. 1996, pp. 62–73.
[44] Kamin III R.A., Adams III G.B., Dubey P.K., "Dynamic trace analysis for analytic modeling of superscalar performance", *Performance Evaluation* **19** (2–3) (Mar. 1994) 259–276.

[45] Kessler R.E., Hill M.D., Wood D.A., "A comparison of trace-sampling techniques for multi-megabyte caches", *IEEE Trans. Comput.* **43** (6) (Jun. 1994) 664–675.

[46] Kessler R.E., McLellan E.J., Webb D.A., "The Alpha 21 264 microprocessor architecture", in: *Proceedings of the 1998 International Conference on Computer Design (ICCD-98)*, Oct. 1998, pp. 90–95.

[47] KleinOsowski A.J., Lilja D.J., "MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research", *Computer Architecture Letters* **1** (2) (Jun. 2002) 10–13.

[48] Lafage T., Seznec A., "Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream", in: *IEEE 3rd Annual Workshop on Workload Characterization (WWC-2000) held in conjunction with the International Conference on Computer Design (ICCD-2000)*, Sep. 2000.

[49] Laha S., Patel J.H., Iyer R.K., "Accurate low-cost methods for performance evaluation of cache memory systems", *IEEE Trans. Comput.* **37** (11) (Nov. 1988) 1325–1336.

[50] Lauterbach G., "Accelerating architectural simulation by parallel execution of trace samples", Tech. Rep. SMLI TR-93-22, Sun Microsystems Laboratories Inc., Dec. 1993.

[51] Loh G., "A time-stamping algorithm for efficient performance estimation of superscalar processors", in: *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS-2001)*, May 2001, pp. 72–81.

[52] Magnusson P.S., Christensson M., Eskilson J., Forsgren D., Hållberg G., Högberg J., Larsson F., Moestedt A., Werner B., "Simics: A full system simulation platform", *IEEE Computer* **35** (2) (Feb. 2002) 50–58.

[53] Manly B.F.J., *Multivariate Statistical Methods: A Primer*, second ed., Chapman & Hall, London/New York, 1994.

[54] Martonosi M., Gupta A., Anderson T., "Effectiveness of trace sampling for performance debugging tools", in: *Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1993, pp. 248–259.

[55] Mauer C.J., Hill M.D., Wood D.A., "Full-system timing-first simulation", in: *Proceedings of the 2002 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Jun. 2002, pp. 108–116.

[56] Michaud P., Seznec A., Jourdan S., "Exploring instruction-fetch bandwidth requirement in wide-issue superscalar processors", in: *Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques (PACT-1999)*, Oct. 1999, pp. 2–10.

[57] Moudgill M., "Techniques for implementing fast processor simulators", in: *Proceedings of the 31st Annual Simulation Symposium*, Apr. 1998, pp. 83–90.

[58] Moudgill M., Wellman J.-D., Moreno J.H., "Environment for PowerPC microarchitecture exploration", *IEEE Micro* **19** (3) (May/June 1999) 15–25.

[59] Mukherjee S.S., Adve S.V., Austin T., Emer J., Magnusson P.S., "Performance simulation tools: Guest editors' introduction", *IEEE Computer, Special Issue on High Performance Simulators* **35** (2) (Feb. 2002) 38–39.

[60] Neefs H., Vandierendonck H., De Bosschere K., "A technique for high bandwidth and deterministic low latency load/store accesses to multiple cache banks", in: *Proceedings of*

the 6th International Symposium on High-Performance Computer Architecture (HPCA-6), Jan. 2000, pp. 313–324.

[61] Nguyen A.-T., Bose P., Ekanadham K., Nanda A., Michael M., "Accuracy and speed-up of parallel trace-driven architectural simulation", in: *Proceedings of the 11th International Parallel Processing Symposium (IPPS'97)*, Apr. 1997, pp. 39–44.

[62] Noonburg D.B., Shen J.P., "Theoretical modeling of superscalar processor performance", in: *Proceedings of the 27th Annual International Symposium on Microarchitecture (MICRO-27)*, Nov. 1994, pp. 52–62.

[63] Noonburg D.B., Shen J.P., "A framework for statistical modeling of superscalar processor performance", in: *Proceedings of the 3rd International Symposium on High-Performance Computer Architecture (HPCA-3)*, Feb. 1997, pp. 298–309.

[64] Nussbaum S., Smith J.E., "Modeling superscalar processors via statistical simulation", in: *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT-2001)*, Sep. 2001, pp. 15–24.

[65] Nussbaum S., Smith J.E., "Statistical simulation of symmetric multiprocessor systems", in: *Proceedings of the 35th Annual Simulation Symposium 2002*, Apr. 2002, pp. 89–97.

[66] Ofelt D.J., "Efficient performance prediction for modern microprocessors", Ph.D. thesis, Stanford University, Aug. 1999.

[67] Ofelt D.J., Hennessy J.L., "Efficient performance prediction for modern microprocessors", in: *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS-2000)*, Jun. 2000, pp. 229–239.

[68] Oskin M., Chong F.T., Farrens M., "HLS: Combining statistical and symbolic simulation to guide microprocessor design", in: *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA-27)*, Jun. 2000, pp. 71–82.

[69] Reilly M., "Designing an Alpha microprocessor", *IEEE Computer* **32** (7) (Jul. 1999) 27–34.

[70] Reilly M., Edmondson J., "Performance simulation of an Alpha microprocessor", *IEEE Computer* **31** (5) (May 1998) 50–58.

[71] Rosenblum M., Bugnion E., Devine S., Herrod S.A., "Using the SimOS machine simulator to study complex computer systems", *ACM Transactions on Modeling and Computer Simulation (TOMACS)* **7** (1) (Jan. 1997) 78–103.

[72] Saavedra R.H., Smith A.J., "Analysis of benchmark characteristics and benchmark performance prediction", *ACM Transactions on Computer Systems* **14** (4) (Nov. 1996) 344–384.

[73] Schnarr E., Larus J.R., "Fast out-of-order processor simulation using memorization", in: *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, Oct. 1998, pp. 283–294.

[74] Sherwood T., Perelman E., Calder B., "Basic block distribution analysis to find periodic behavior and simulation points in applications", in: *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques (PACT-2001)*, Sep. 2001, pp. 3–14.

[75] Sherwood T., Perelman E., Hamerly G., Calder B., "Automatically characterizing large scale program behavior", in: *Proceedings of the 10th International Conference on Ar-*

chitectural Support for Programming Languages and Operating Systems (ASPLOS-X), Oct. 2002, pp. 45–57.

[76] Skadron K., Ahuja P.S., Martonosi M., Clark D.W., "Branch prediction, instruction-window size, and cache size: Performance tradeoffs and simulation techniques", *IEEE Trans. Comput.* **48** (11) (Nov. 1999) 1260–1281.

[77] Smith J.E., Sohi G.S., "The microarchitecture of superscalar processors", *Proceedings of the IEEE* **83** (12) (Dec. 1995) 1609–1624.

[78] Sorin D.J., Pai V.S., Adve S.V., Vernon M.K., Wood D.A., "Analytic evaluation of shared-memory systems with ILP processors", in: *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA-25)*, Jun. 1998, pp. 380–391.

[79] Squillante M.S., Kaeli D.R., Sinha H., "Analytic models of workload behavior and pipeline performance", in: *Proceedings of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS-1997)*, Jan. 1997, pp. 91–96.

[80] Srivastava A., Eustace A., "ATOM: A system for building customized program analysis tools", Tech. Rep. 94/2, Western Research Lab, Compaq, Mar. 1994.

[81] StatSoft Inc., "STATISTICA for Windows. Computer program manual", http://www.statsoft.com, 1999.

[82] Sugumar R.A., Abraham S.G., "Efficient simulation of caches under optimal replacement with applications to miss characterization", in: *Proceedings of the 1993 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'93)*, 1993, pp. 24–35.

[83] Tomasulo R.M., "An efficient algorithm for exploiting multiple arithmetic units", *IBM Journal* **11** (Jan. 1967) 25–33.

[84] Uhlig R., Nagle D., Mudge T., Sechrest S., Emer J., "Instruction fetching: Coping with code bloat", in: *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA-22)*, Jun. 1995, pp. 345–356.

[85] Wood D.A., Hill M.D., Kessler R.E., "A model for estimating trace-sample miss ratios", in: *Proceedings of the 1991 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1991, pp. 79–89.

[86] Wunderlich R.E., Wenish T.F., Falsafi B., Hoe J.C., "SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling", in: *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA-30)*, Jun. 2003.

[87] Yeager K.C., "MIPS R10000 superscalar microprocessor", *IEEE Micro* **16** (2) (Apr. 1996).

[88] Yi J.J., Lilja D.J., Hawkins D.M., "A statistically rigorous approach for improving simulation methodology", in: *Proceedings of the 9th International Symposium on High Performance Computer Architecture (HPCA-9)*, Feb. 2003.

Security Issues and Solutions in Distributed Heterogeneous Mobile Database Systems

A.R. HURSON, J. PLOSKONKA,
Y. JIAO, AND H. HARIDAS

Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802
USA
hurson@cse.psu.edu
ploskonk@cse.psu.edu
yjiao@cse.psu.edu
haridas@cse.psu.edu

Abstract

Security continues to be a fundamental requirement of modern computer systems. As more information is stored electronically, users become more concerned with the security of their information. Designing secure systems is a difficult problem, complicated by the distributed nature of modern systems—communication links are now both wired and wireless and information is shared among autonomous and heterogeneous sources. Once the system is deployed, security threats and attacks must be evaluated and handled throughout its life cycle, as attackers discover and exploit vulnerabilities. This chapter considers the security problem within the context of centralized databases and multidatabases as well as mobile systems. The techniques employed to create a secure centralized database system can be extended to meet the security needs of multidatabase systems. This extension is not trivial, however. Many issues must be considered so that the needs of each component are met. When wireless communication links and mobility are introduced, adapting existing methods may not provide a satisfactory solution. Generally, security is expensive in terms of communication, processing, and storage overhead, which translates into increased power consumption. This overhead is unacceptable in mobile systems because mobile devices are already resource-poor. Therefore, new techniques, which draw from existing technologies and the lessons learned from their deployment, must be designed to handle the constraints introduced by mobile systems.

ISSN: 0065-2458/DOI 10.1016/S0065-2458(03)61003-X
Copyright © 2004 Elsevier Inc.
All rights reserved.

1. Introduction and Motivation

Users have been demanding information "anytime, anywhere." The diversity in the range of information accessible to a user at any time is growing rapidly. Accessing diverse and autonomous information repositories with different APIs (Application Program Interfaces) is not accepted since the user has to be retrained to "*learn a new API and its usage on accessing different information sources*." As a result, a need to integrate diverse information from different sources and provide a user with common APIs has become a necessity. Adding further complexity, the user mobility and privacy concerns prevent gaining knowledge about exact user location, which otherwise could aid in reducing the access latency to the required information repository. Thus, APIs need to:

- Locate the requested information transparently and intelligently based on the user's security level,
- Access, process, and integrate information repository/repositories intelligently and efficiently based on the user's access right,

- Locate the current location of the user efficiently without violating user's privacy,
- Reliably transport resultant information to the user efficiently and securely, and
- Display the information to the user according to his/her display category.

It should be reemphasized that the information requested might not be confined to a single information repository and distributed across numerous repositories at different geographic locations. Hence, in some cases, additional phases of decomposition of requests and integration of intermediate results would be required.

To gain further insight regarding the global information processing, APIs, and limitations of infrastructure several related issues and their possible solutions are studied. The information repositories could include legacy systems, database systems, data warehouses, information services (i.e., stock quotes, news, airline information, weather information, etc.), and the almost limitless information available on the Internet and the World Wide Web [76]. The APIs include traditional systems to access data in the form of relational, object-oriented, distributed, federated, or multidatabase management systems. The expanding technology is making available a wide breadth of devices to use these APIs. Potential devices include desktop computers, laptops, PDAs (Personal Digital Assistants), and cellular phones. The access mechanisms vary according to the access devices and the environments in which data is being accessed. For example, data can be accessed from a desktop workstation connected to a LAN (Local Area Network), or from a laptop computer via a modem, or from a PDA via a wireless communication medium. Access devices have different memory, storage, network, power, and display requirements [76].

Within the scope of traditional multidatabases, as a solution to the integration of information from a collection of autonomous and heterogeneous sources, researchers have effectively addressed issues such as local autonomy, heterogeneity, transaction management, concurrency control, transparency, and query resolution in a "sometimes, somewhere" environment. Consequently, the aforementioned issues and solutions are defined based on fixed clients and servers connected over a reliable network infrastructure. A mobile data access system (MDAS) relies on the information stored in multidatabases. A MDAS is distinguished from a multidatabase by the communication links between nodes. Clients (and possibly even servers) are no longer fixed—they are mobile and can access data over wireless links. Client mobility may apply in both idle (not actively accessing the system) and active modes. The concept of mobility, however, introduces additional complexities and restrictions in multidatabase systems. A user accessing data through a remote connection with a portable device has a high probability of facing problems such as reduced capacity network connections, frequent disconnections, and processing and resource restrictions. Research on providing solutions in a wireless medium is underway. The trade offs are

being identified gradually and prototypes accordingly are being developed, tested, and commercialized.

Some current systems have the means to provide timely and reliable access to remote[1] data in wired and wireless media. However, this alone is not sufficient to ensure content users. Users desire secure communication sessions. Confidentiality, integrity, and authenticity are required to meet user's security expectations. Within the scope of the global information sharing process, the security aspect has received less attention [66,71,73,75]. Trust is a major factor that needs to be established when creating a secure environment that allows remote data access to users. Violating trust would result in havoc in the system. Providing security in distributed systems is, however, a difficult task. The wireless medium further imposes lower bandwidth, frequent disconnections, higher error rates, and non-secure links in the face of autonomy and heterogeneity [65]. Site autonomy is a measure of enforcing local security by the local administrator. Whether security is enforced globally and/or locally (due to local autonomy), a secure global information system must address issues such as access control, authorizations and counter-measures for inferential security, and accountability. Users should be given access to information based on their role or access rights in the global system. Pervasive computing systems (Section 4.5), which are just beginning to appear, extend the scope of MDAS. The security issues in both multidatabases and MDAS still apply, but are further complicated by user privacy concerns and resource-poor devices.

The remainder of this chapter is organized in the following manner. Section 2 looks at security issues in centralized database systems. This discussion provides the reader with the necessary background information on security methods, which are then discussed within the context of multidatabases and Mobile Data Access Systems (MDAS). Section 3 provides an overview of multidatabase systems in general before describing security issues and solutions in further detail. The impact mobility has on these issues is considered in Section 4, which also gives a brief overview of pervasive computing systems and the security threats that must be considered. Finally, Section 5 summarizes the key points of this chapter and briefly discusses some future work in related areas.

2. Security Issues of Centralized Database Systems

Three interrelated technologies are used to achieve information confidentiality and integrity in traditional DBMSs: authentication, access control, and audit [1]. Figure 1

[1] Remote access to data refers to both mobile nodes and fixed nodes accessing a variety of data via network connection characterized by (1) lower bandwidth, (2) frequent disconnection, and (3) higher error rates [76].

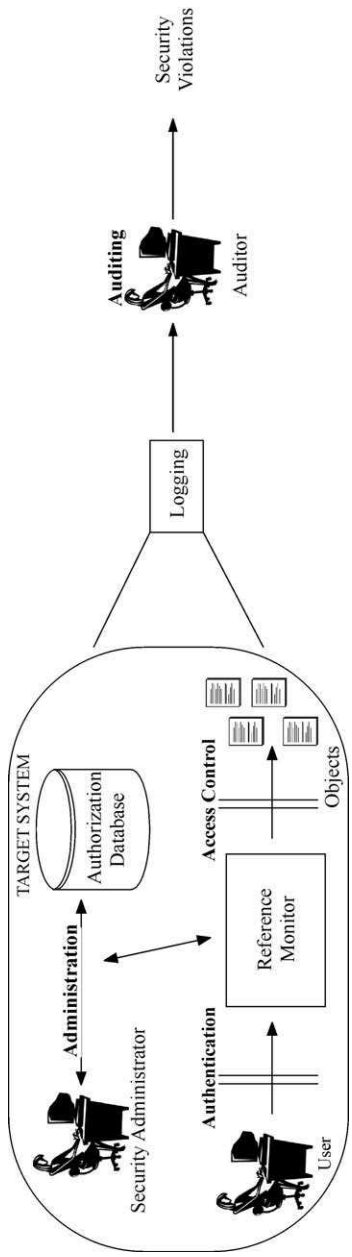


FIG. 1. Access control and other security services [1].

logically illustrates how these three services interact with each other to ensure the security of the DBMS. Authentication identifies one party to another and it can be performed in both directions. It ensures the true identity of a user, computer, or process. Once the identity is established, the party can access data under the guidance of the access control service. Access control regulations are set by the system security administrator and define who can access what data with which privileges. However, authentication and access control do not comprise a complete security solution—they must be complemented by an auditing service. An auditing service records important user activities in system logs for real-time or a posteriori analysis. Real-time auditing is often referred to as intrusion detection. An audit service protects a system in three ways: detecting actual security violations; assisting the security administrator in discovering attempted attacks by recognizing abnormal activity patterns; and detecting possible system security flaws.

These three security services support and complement one another. In the following subsections we define some important terminology and then discuss authentication, access control, and auditing in further detail.

2.1 Terminology and Notation

- *Plaintext* is a message in its unencrypted form, either before the encryption transformation has been applied, or after the corresponding decryption transformation is complete [2].
- *Ciphertext* is the encrypted form of a message—the output of the encryption transformation [2].
- *Encryption* is the process of transforming data into a form that cannot be understood without applying a second transformation. The transformation is affected by an encryption key (E_K) in such a manner that the second transformation can only be applied by someone in possession of the corresponding decryption key (D_K) [2].
- A *secret-key cryptosystem* (*or symmetric cryptosystem*), such as that defined by the Data Encryption Standard (DES) [3], uses a single key for both encryption and decryption ($E_K = D_K$). Such an encryption key is called a *secret key* ($K_{A,B}$).
- A *public-key cryptosystem* (*or asymmetric cryptosystem*), such as RSA [4], uses different keys for encryption and decryption ($E_K \neq D_K$). One of the keys in the pair can be publicly known while the other must be kept private. These keys are referred to as *public and private keys* (K_A^+ and K_A^-), respectively.
- The *spoofing attack* refers to the situation where one computer masquerades as another.

TABLE I
NOTATION USED IN THIS CHAPTER

Notation	Description
E_K	Encryption key
D_K	Decryption key
$K_{A,B}$	Secret key shared by A and B
K_A^+	Public key of A
K_A^-	Private key of A
$E(K, r)$	Encrypt r with key K
$D(K, r)$	Decrypt r with key K

- The *brute force attack* (or, exhaustive key space attack) cracks the password by performing an exhaustive search over the entire key space.
- The *dictionary attack* tries a list of words, possible weak passwords, and their simple transformations, such as capitalizing, prefixing, suffixing or reversing a word, until the hashed value of the candidate matches a password hash [5].

2.2 Authentication

Authentication identifies one party to another, for instance, identifying a user to a computer system. It serves as the foremost guard of the whole security system. Without a reliable authentication mechanism, strong access control and intrusion detection seem pointless. The most commonly deployed authentication mechanism is password authentication. When a user logs on to a computer system by entering the user identifier and password, that user is identified to the computer system.

Authentication is often required in both directions. For example, in a peer-to-peer network, peer computers need to identify each other. In a client-server environment, the client should identify itself to the server in order to obtain service. Authenticating the server to the client is also important to prevent spoofing attacks and protect sensitive client information.

Authentication mechanisms are often based on one or more of the following techniques:

- Knowledge-based authentication,
- Token-based authentication,
- Biometric-based authentication, and
- Recognition-based authentication.

2.2.1 Knowledge-Based Authentication

Knowledge-based authentications use "something that you know" to identify you. Because it is easy to implement and has the advantage of low cost, the password is the most common mechanism used in knowledge-based authentications. However, research has shown that it has significant security deficiencies [1]. Several well-known problems associated with passwords are as follows:

- Passwords can be discovered by social engineering because users tend to select passwords that are easy to remember such as birth dates, names, social security numbers, phone numbers, etc. Unfortunately, such passwords can be easily guessed by password crackers. Problems caused by allowing users to choose passwords without restriction have been discussed in [6,7].
- Passwords can be snooped by observing the user entering it.
- Password sharing is another intrinsic problem of this technique.
- The growing use of Internet applications has led to users having many accounts. Managing a large number of user ID/password combinations is difficult, causing users to either use the same password for every account or write them down.

Password management is required to prod users to regularly change their passwords, select good ones, and protect them with care. Five basic solutions have been proposed in the literature: user education, random password generation, reactive password checking, proactive password checking, and one-time passwords [6,8–11].

2.2.1.1 User Education. This method educates users to choose strong passwords. However, it is very difficult in environments where there are a significant number of novices. First, users might not understand the importance of choosing strong passwords, and novice users are not the best judges of what is strong. For instance, novice users (mistakenly) may believe that reversing a word, or capitalizing the last letter makes a password strong. Second, education may backfire. If the education provides users with a specific way to create passwords, such as using the first several letters of a favorite phrase, many of the users may use that exact method and thus, make an attack easier.

2.2.1.2 Random Password Generation. Random password generation intends to eliminate weak password choices by providing users with randomly generated passwords from the whole key space [9]. This seems to be a perfect solution to strengthening passwords, however, it also has flaws. First, the random mechanism might not be truly random, and could be analyzed by an attacker. Second, random passwords are often difficult to memorize (user-unfriendly). As a result, users may write the passwords down. This provides attackers an opportunity to obtain

passwords effortlessly. Furthermore, secure distribution of such passwords is a difficult task. To make randomly generated passwords more user-friendly, some systems generate pronounceable passwords for users. However, these systems have also been shown to be insecure [12].

To overcome the shortcomings of randomly generated passwords, researchers proposed two alternate methods that provide supervision of user-chosen passwords, namely reactive password checking and proactive password checking. Interestingly, the basis of current password checkers is their enemy—the dictionary attack. When cracking passwords, hackers may use the dictionary attack or brute force attack (see Section 2.1). Hackers usually favor the former over the latter because it can be accomplished in hours using commonplace workstations, while the brute force attack sometimes is computationally infeasible. For the same reason, the current password checkers are based on the dictionary attack method. The checker compares the user-chosen password against a dictionary of weak passwords. If a candidate password matches a dictionary item or its common transformations, the candidate password will be considered as a weak password. Reactive password checking and proactive password checking methods invoke the checker at different phases of the authentication process and therefore, demonstrate different characteristics.

2.2.1.3 Reactive Password Checking. Reactive password checking prevents poor password choices by performing a posteriori password scan. The algorithm, the scanning program, is the same as the dictionary attackers. If the scanning program successfully cracked a user's password, the password is then considered to be weak and the user will be notified to change his/her password. Systems that support reactive password checking include *deszip* [13] and COPS [14]. There are significant problems with this approach [10].

- The dictionary used by the scanner may not be comprehensive enough to catch all weak passwords.
- Since the scanning process takes place after passwords have been chosen, a lucky attacker may be able to crack some weak passwords before the scanning program detects them. This is especially problematic when the number of users is large.
- The output of a scanner may be intercepted and used against the system.
- Finding a weak password does not guarantee that it will be replaced by a strong one.

2.2.1.4 Proactive Password Checking. Proactive password checking is stemmed from the same idea as the reactive password checking. The major difference is that the weak password scanning process takes place before a password is

recorded as legitimate. By disallowing the choice of poor passwords in the first place, proactive password checking avoids some of the drawbacks that reactive password checking exhibits.

The proactive password checking, however, also has its difficulties. In particular, as the size of the dictionary grows, in order to achieve acceptable response time, a proactive password checker must seek algorithms that support both fast checking and effective compression. For example, the OPUS system [10] uses Bloom filters [15], the Bapasswd system [16] uses trigrams and a Markov model, and the ProCheck system [17,18] applies a decision-tree technique to achieve high dictionary compression (up to 1000 : 1) and fast checking speed.

2.2.1.5 One-Time Passwords.

As pointed out by [11], the source of many difficulties in password-based authentication is the fundamental premise—the reusability of passwords. One-time passwords can substantially reduce the threat of password crackers. For example, if a password is a six-digit number that changes randomly minute by minute, the attacker must perform a million login attempts in a minute to penetrate the security system. This is impossible considering that the maximum number of login requests a system can handle per minute is a few hundred [11]. Attempts to crack into the system by observing a user keying in the password will also fail because the system will not accept reused passwords. Because a new random password must be generated frequently, systems that support one-time passwords usually need the assistance of extra hardware such as SmartCards (often referred to as tokens), which we will introduce in the next subsection.

2.2.2 Token-Based Authentication

Token-based authentication use "something that you possess" to identify you, such as a SmartCard. Technically, SmartCards are credit card-sized cards with more memory than the traditional magnetic strip and an on-board embedded processor, or smart chip. Since people in general are used to carrying keys, credit cards, etc., it is expected that using SmartCards does not introduce significant inconvenience. In this subsection, we will briefly review the evolution of SmartCards, present an overview of different types of SmartCard, and finally discuss two SmartCard-based authentication protocols.

2.2.2.1 A Brief History of the SmartCard.

The SmartCard made its debut in Germany in 1968. Two German inventors patented the idea of embedding microchips in plastic cards [19]. The Japanese patented another version of the SmartCard in 1970 [20]. In 1974, Roland Moreno invented IC card (a type of SmartCard) in France. Later, he received a patent in France in 1975 and in the US in 1978.

Even though the idea of the SmartCard originated in the late 1960s, the technology that could support such an innovative idea was not available until 1976 [19]. In 1977, Motorola Semiconductor, in conjunction with Bull, a French computer company, produced the first SmartCard microchip [21]. Since then, the SmartCard has been deployed in a wide range of applications [22].

2.2.2.2 Types of SmartCards. A SmartCard can be categorized as either a memory card or a processor-enabled card (see Table II) [22]. A memory card provides nonvolatile storage of personal information and it does not have data processing capability. The capacity of memory cards is usually much higher (several KB) than traditional magnetic strip-based cards (hundreds of bytes). Information stored on a memory card is read-only: the information flows from the card to the card reader, but not vice versa. Sometimes memory cards are referred to as "asynchronous cards" because they are used offline.

As semiconductor technology advances, processor-enabled SmartCards appeared on the horizon [23]. Compared to memory cards, processor-enabled SmartCards are equipped not only with read-only memory (ROM), but also with a CPU and random access memory (RAM). Data stored on a SmartCard can now be protected by cryptography technologies. These cards are sometimes referred to as "synchronous cards" as the data flow is bi-directional: data is read from and written to the card [19]. More advanced processor-enabled SmartCards can even support downloading of new applications. Theses cards are more expensive and often called "white cards" [24].

2.2.2.3 SmartCard-Based Authentication. We have discussed the advantages of one-time passwords in the previous section. We also mentioned that such one-time passwords could be easily implemented on SmartCards. Processor-

TABLE II
MEMORY VS. PROCESSOR-ENABLED SMARTCARDS [22]

Feature Component	SmartCard	
	Memory Card	Processor-Enabled Card
Read-Only Memory?	Yes	Yes
Random Access Memory?	No	Yes
Microprocessor?	No	Yes
Contact/Contactless Interface	Contact, contactless, or both	Contact, contactless, or both
Data certified secure (ITSEC[a])?	No	Yes
Example	Phone card	Multi-application cards

[a]Information Technology Security Evaluation Certification represents a set of software and hardware security standards that have been adopted in Europe and Ausralia.

enabled SmartCard technology is now mature and cheap enough to support one-time password protocols and strengthen the power of authentication mechanisms.

The challenge-response and time-series authentication protocols are two examples of token-based authentication schemes [11]. They both rely on the cryptographic capability of SmartCards. Tokens can use either secret key (e.g., DES) or public key (e.g., RAS) cryptosystems. Figures 2–5 illustrate these two protocols utilizing different cryptosystems. These figures follow the notation convention shown in Table I and assume that user **A** is trying to log on to the system. Please also note that the following protocols assume a centralized authentication environment and therefore, information that flows between user **A** and the **AS** (Authentication Server) cannot be eavesdropped by attackers. When moving to a distributed environment, extra measurements must be taken to prevent replay attacks etc. We will consider those issues in Section 3.

Figure 2 depicts a challenge-response protocol that uses a secret-key cryptosystem: user **A** and the Authentication Server (AS) share key $K_{A,AS}$. The assumption is that the shared key has been securely stored in both **A**'s token and the **AS**. When **A** sends a log in request to the **AS**, the **AS** first looks up its user-key pair table to find out the shared key $K_{A,AS}$. It then sends a randomly generated number r (a challenge) to **A** and waits for a response. Upon receiving r, **A** keys the number into the SmartCard using the keypad. The SmartCard encrypts this number using the shared key $K_{A,AS}$. The encryption result is displayed on the SmartCard, 837 269 in this ex-

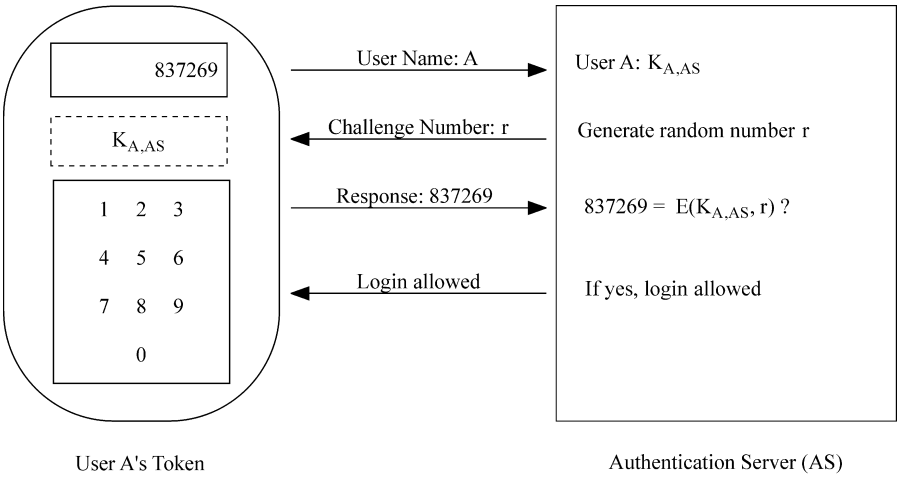


FIG. 2. Challenge-response protocol using secret-key cryptosystem.

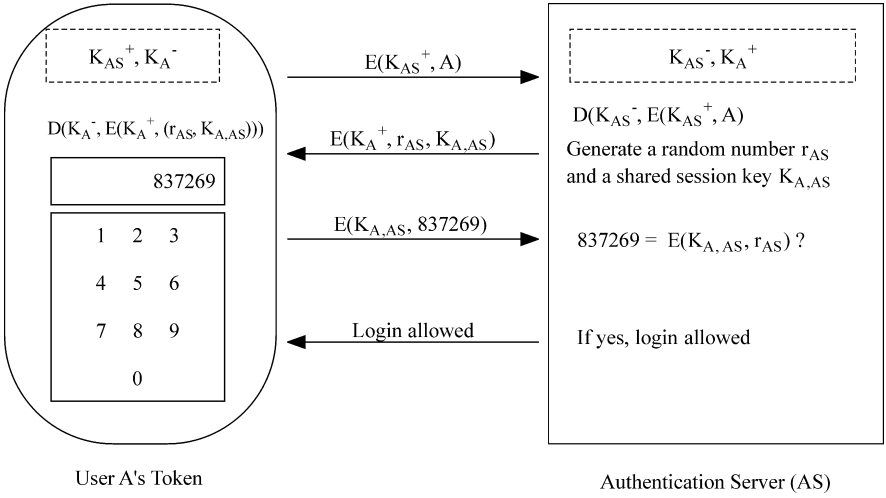


FIG. 3. Challenge-response protocol using public-key cryptosystem.

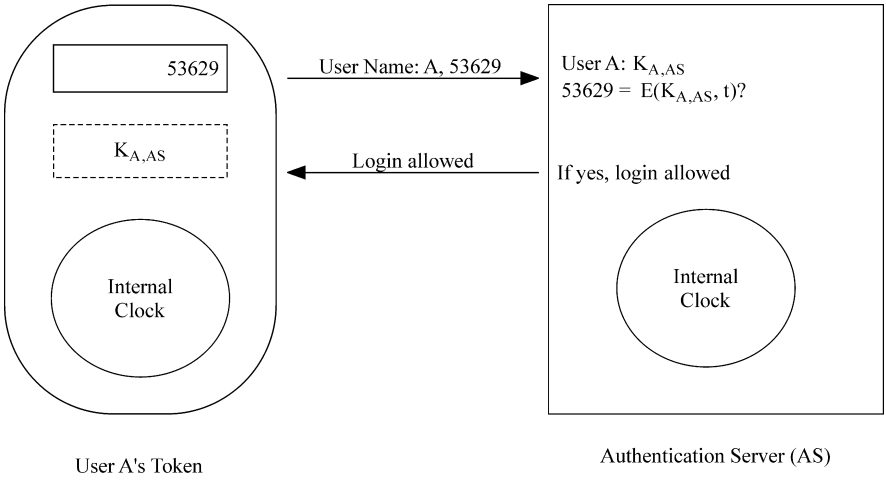


FIG. 4. Time-series protocol using secret-key cryptosystem.

ample. **A** sends this result to the **AS** as the response to its challenge. If the response from **A** matches the number calculated by the **AS** using the same shared secret key and encryption algorithm, user **A** is authenticated.

Figure 3 shows a challenge-response protocol that uses a public-key cryptosystem. User **A** and the **AS** know about each other's public key and their own private keys. User **A** initiates a login session by sending his/her user name encrypted by the **AS**'s public key. Once this message is decrypted, the **AS** generates a random number r_{AS} (a challenge) and a shared session key $K_{A,AS}$ and sends them to the login request originator **A** (encrypted by A's public key). After user **A** obtains the shared session key, the response is calculated and verified in the same manner as described in Fig. 2.

A time-series protocol utilizing a secret-key cryptosystem assumes that the **AS** has all potential user's secret keys and each user's SmartCard has a built-in clock that is well synchronized with that of the **AS**. This protocol simplifies the authentication procedure by applying the synchronized clock time t as the default challenge number for the user's token. The time value t is enciphered by using the user's secret key $K_{A,AS}$ and displaying it on the SmartCard at a preset frequency, e.g., every minute. In this approach, all users get the same challenge number because all internal clocks throughout the system are synchronized. However, the response generated by each SmartCard is different since different keys encipher the same challenge.

At a particular time if user **A** wants to login to the system, he/she enters the user name and the number currently displayed on his/her SmartCard. The **AS** authenticates the user by comparing the response (53 629 in Fig. 4) entered by the user with the number generated by enciphering its own clock time with the secret key of that user. If the two numbers match, the login permission is granted to user **A**.

Figure 5 illustrates a similar protocol as the one shown in Fig. 4. The difference is that a public-key cryptosystem is used in this case. The authentication procedure starts with user **A** sending his/her user name encrypted by the **AS**'s public key to the **AS**. The **AS** then generates a shared session key $K_{A,AS}$ and sends it back to user **A**. Note that this session key is encrypted by **A**'s public key K_A^+. The rest of the authentication is the same as we described in Fig. 4, except replacing the secret key of **A** in Fig. 4 with the shared session key in Fig. 5.

Tokens that use secret or public key cryptosystems have their own advantages and disadvantages. Secret-key cryptosystem is much faster than public-key cryptosystem. However, the secret key distribution is problematic. Moreover, the public-key cryptosystem based authentication systems have better scalability because the authentication server does not need to maintain a key for each potential user as in a secret key cryptosystem.

Token-based authentication is much stronger than password-based authentication, and it is often called strong authentication [1]. However, tokens are subject to theft or sharing. Extra effort must be made to protect tokens. For instance, SmartCards can be protected by PINs (Personal Identification Numbers): A PIN number is required to activate the token. If a SmartCard were stolen, it would take more time for a thief to discover the PIN than for a system administrator to disable the card.

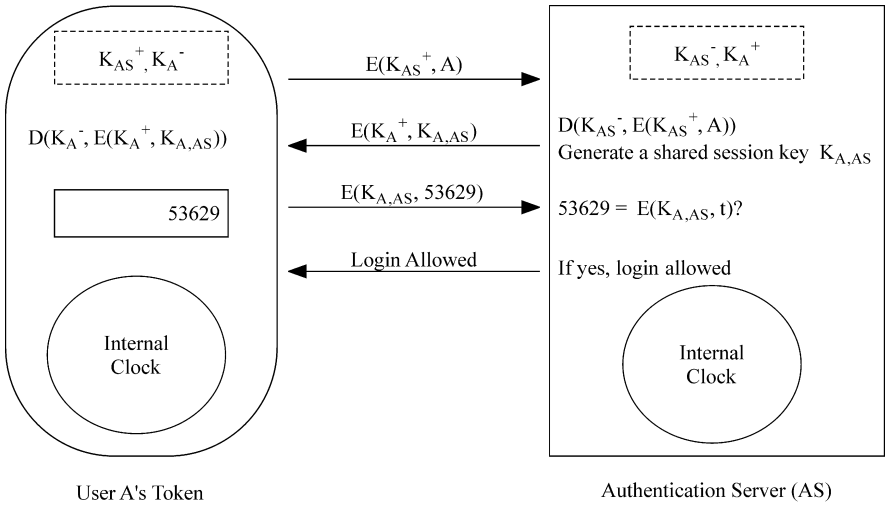


FIG. 5. Time-series protocol using public-key cryptosystem.

2.2.3 Biometric-Based Authentication

Biometric-based authentication uses "your physiological or behavioral characteristics" to identify you, such as fingerprint, voice, etc. In general, any physiological or behavioral characteristic that is universal, unique, permanent, and collectable could be used as a biometric [25].

One problem that occurs in both knowledge-based and token-based authentication systems is that the information needed may not be always available: passwords may be forgotten and tokens may be left at home. In addition, these systems are unable to differentiate between an authorized person and an impostor who knows the correct password or possesses the proper token. An alarming number was reported by the banking industry [26]: The false acceptance rate at Automatic Teller Machines (ATM) is as high as 30% and results in worldwide financial fraud of $2.98 billion a year.

Biometric techniques have a long history in criminal identification. As the technology rapidly evolves and the price drops, biometric-based authentication is becoming a powerful tool for a broad range of civilian applications. It can be used for two different purposes: biometric identification and biometric verification [27]. Biometric identification (recognition) identifies a person from the database of persons known to the system and answers the question "Who am I?" Biometric verification authenticates a claimed identity and answers the question "Am I who I claim I am?"

2.2.3.1 Biometric System. A biometric-based authentication system may deploy one or more of the biometric technologies: voice recognition, fingerprints, face recognition, iris scan, infrared facial and hand vein thermograms, retinal scan, hand and finger geometry, signature, gait, and keystroke dynamics [25]. Table III summarizes the most common biometric systems.

The authentication process is usually performed in two phases:

(1) *Enrollment phase*. During this phase, the biometric authentication system scans a user's biometric information into the system and transforms it into

TABLE III
BIOMETRIC TECHNOLOGIES

Technology	Based on	Advantages	Disadvantages
Facial Recognition [28]	The distance between facial attributes or on the dimensions of the attributes themselves	• Acceptable to users • Natural & intuitive • Inexpensive • Works well under constrained conditions	• Sensitive to variations in illumination and faces with different positions or expressions • Performs poorly as database size increases • Identical twins are hardly distinguished
Facial Thermograms	The blood vessel pattern of the face	• Acceptable to users	• May not be discriminative enough with large databases • Can be sensitive to the emotional state of the subject or body temperature
Fingerprints	Fingerprints	• Unique • Distinct even in identical twins • Do not change over time	• Dry skin & dirt can affect performance • May not be usable due to cuts or scars • Associated with identifying criminals • Requires a large amount of computational and storage resources
Hand Geometry	The shape of the hand (both top & sides)	• Requires less storage than fingerprints • Less expensive than fingerprints • More acceptable to users than fingerprints	• Performance can depend on weather conditions or hand cleanliness • Shape of hand may change over a lifetime • Tests with simulated hands have not been performed • Physical size of sensor is large—not suitable for certain applications

(*continued on next page*)

TABLE III — *Continued from previous page*

Technology	Based on	Advantages	Disadvantages
Iris Scan [29]	The colored ring that surrounds the pupil	• After DNA, irises are the most individualized feature of human body • Does not require contact between subject's eye and biometric device • Inexpensive • Irises are less susceptible to injury than other body parts • Template requires only a few bytes • Works even if subject is wearing glasses	
Retinal Scan	The pattern of veins in the eye		• Users must stand close to the device and focus on a target • Sensors are expensive • Retinas change over a lifetime
Voice Recognition [30]	Voice	• Favored by users • Can provide access to secure data over telephone lines	• Speech patterns change due to stress and disposition • Background noises and quality of telephone connection can reduce performance • Twins (siblings, to a lesser extent) are more difficult to distinguish than the general population
Signature recognition	Each person's unique style of handwriting	• Acceptable to users • Natural and intuitive	• Two signatures are never exactly the same • Not reliable for large populations • Expensive

digital representation (called a template). The digital form of the user's biometric information is then stored in a database.

(2) *Identification phase.* In this phase, a user's biometric information is collected and digitized in the same way as in the enrollment stage. This digital representation of the user is compared to the templates stored in the database to establish the identity of the user. Depending on the matching result, the user will be either accepted or rejected.

2.2.3.2 Performance Measurement. Traditionally, two error statistics are often used in biometric authentication system evaluation [31]:

- *False Accept Rate* (FAR) (sometimes called False Match Rate), the percentage of impostors accepted, and
- *False Reject Rate* (FRR) (sometimes called False Non-Match Rate), the percentage of authorized users rejected.

The performance of a biometric authentication system is measured in terms of the FRR achieved at a fixed FAR, or vice versa [32]. By varying the FAR value, a so-called receiver operating characteristic (ROC) curve is obtained. On the ROC curve, the point where FRR = FAR is called the equal error rate (EER). It is used to describe the performance of the overall system. The better biometric systems have EERs of less than 1%.

2.2.3.3 Common Characteristics of Biometric Systems. Table IV identifies the major characteristics of a biometric system [27].

2.2.3.4 Privacy Issues. The issue of privacy is controversial and essential in biometric systems. Because the biometric information gathered by the system is unique and permanent, users are naturally extremely concerned about the safety and proper use of such information. Without a guarantee of user privacy, it is hard to imagine that the public will accept a biometric system.

Many researchers are trying to resolve the privacy problem. In [33], the author suggests a solution that combines biometric-based authentication and token-based authentication. The solution is to equip people with personal devices (called wal-

TABLE IV
COMMON CHARACTERISTICS OF BIOMETRIC SYSTEMS

Uniqueness	• The biometric system must be based upon distinguishable personal traits (i.e., no two persons should be the same in terms of the characteristic the system is based on).
Universality	• Each person should have the characteristic the system is based on.
Permanence	• The characteristic should neither change nor could be altered during lifetime.
User-friendliness	• The system must be easy to use and not intrusive.
Cost	• Compared to knowledge-based and token-based authentication systems, the cost of biometric systems can become a major factor that deters the wide acceptance of such systems.
Accuracy	• Depending on the application, a biometric system needs to achieve an appropriate balance between the False Accept Rate and False Reject Rate.

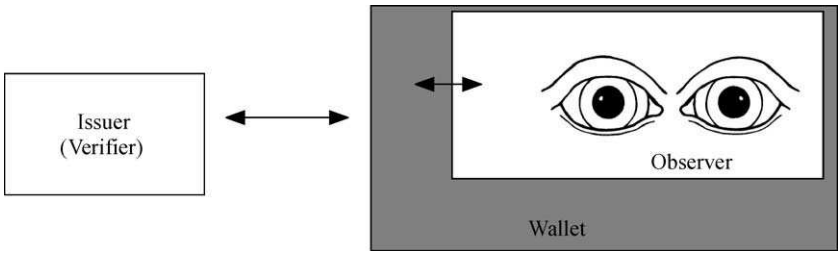


FIG. 6. Wallet with observer architecture [33].

lets) that can run a trusted local biometric verification process (called observer). The system is depicted in Fig. 6.

The function of the observer is to verify that the person holding the wallet is the correct owner. When the wallet is issued, the observer is personalized (and cannot be re-personalized) for the particular biometric identity. When the issuer wants to authenticate the wallet holder, authentication is based only on the wallet (token-based approach). However, the wallet is only activated if the observer successfully authenticates the user's biometric identity. This architecture protects user privacy by preventing inflow and outflow of information from the observer to the issuer.

From the discussion in this section, we conclude that biometric-based authentication has many unique merits. However, it is not the panacea of authentication. Table V lists the major advantages and disadvantages of biometric-based authentication [27].

2.2.3.5 Examples of Biometric Applications. Currently, both the public and private sectors are making extensive use of biometrics. The following list of current real world application, though not exhaustive, gives an idea of the potential of this new approach to identity authentication (see Table VI).

2.2.4 Recognition-Based Authentication

Recognition-based authentication uses "what you can recognize" to identify you. The foundation of this technique is that studies have shown that it is much easier to recognize something than to recall the same information from memory without help [37]. Image recognition is the favored recognition-based authentication system because classic cognitive science experiments show that humans have a vast, almost limitless memory for pictures [38,39]. By replacing precise recall of a password with image recognition, it is expected that the user's cognitive load will be reduced and therefore, the system can provide a more pleasant user experience. Much research and implementation efforts have been reported in the literature.

TABLE V
ADVANTAGES AND DISADVANTAGES OF BIOMETRIC-BASED AUTHENTICATION

Advantages	Disadvantages
• A biometric provides **positive identification** of the individual who performed a given transaction.	• **Lack of a single standard** supported by the entire industry. For the shop owner, the diversity of biometric methods would require perhaps a dozen data collection devices at every cash register. Even if a single biometric measure such as fingerprinting were accepted as the standard, there are dozens of proprietary formats for data storage and analysis.
• It **enhances user service** by providing quick and easy identification.	• Most of the biometric technologies work well only for a **small target population**.
• A biometric identification system is **based on something that cannot be stolen** or compromised.	• The level of **public concern** about privacy and security is high.
• It **requires no teller or operator interpretation**. Biometric identification depends on computer algorithms to make a yes/no decision.	• Biometric technologies **do not fit well in remote systems**. If the verification takes place across a network (the measurement point and the access control decision point are not co-located), the system might be insecure. This problem can be overcome by the use of a secure channel between the two points.
	• Biometric systems **do not handle failure well**. If someone steals ones bio-information, it remains stolen for life.

Blonder [40] patented a "graphical password." This approach requires a user to touch an image on predetermined regions in a predefined sequence. Only when both the touched regions and the sequence are correct, the user is said to be authenticated. As we can see, this method still requires precise recall. Moreover, it is vulnerable to observer attacks.

Jermyn et al. [41] proposed a graphical password selection and input scheme similar to [40], where the password consists of a simple picture drawn on a grid. This solution removes the need for temporal recall, but still requires users to precisely recall how to draw their image, rather than relying on recognition.

IDArts [42] designed an authentication system based on recognizing previously viewed images of faces, called Passfaces. A drawback of the system is that users choose faces that they are attracted to and therefore, can be easily guessed by attackers.

Rachna Dhamija and Adrian Perrig designed the Déjà Vu authentication system using random art images [43,44]. The user chooses a set of p images as his/her portfolio. At the authentication point, the user is presented with a mixture of portfolio

TABLE VI
EXAMPLES OF BIOMETRIC APPLICATIONS

Location/Date Implemented	Technology	Purpose
University of Georgia at Athens 1994 [34]	Hand recognition	• Restrict cafeteria access to students enrolled in its meal plan
Colombian Legislature 1992	Hand geometry	• Confirm identity of the members of its two assemblies immediately before a vote
John F. Kennedy and Newark International Airports INSPASS (Immigration and Naturalization Services Passenger Accelerated Service System) 1993 [35]	Hand geometry	• Verify identity of frequent travelers to the US
US/Canada border PORTPASS [35]	Voice recognition	• Monitor vehicle occupants
Walt Disney World, Orlando, Florida [26]	Finger geometry	• Season passes
Woolworth's supermarket, Australia [26]	Fingerprints	• Monitor time and attendance for 100,000 employees
Langkawi Airport, Malaysia 1997	Face recognition	• Reconcile passengers with their luggage from check-in to boarding to prevent terrorists from checking luggage and not boarding the plane
Charlotte/Douglas International Airport, North Carolina; Flughafen Frankfurt Airport, Germany [36]	Iris scan	• Identify airport employees
Bank United Corporation, Houston 1999 [36]	Iris scan	• Identify supermarket customers before they conduct transactions at ATMs

images and decoy images. The user must recognize all portfolio images to be authenticated. Figure 7 shows an example of the image selection window. This system resists many potential attacks well. However, it is still hard to justify that the images cannot be guessed by knowing a specific user's taste of images.

2.2.5 Summary

Table VII summarizes different authentication methods as discussed in this chapter. As one can conclude, technically the best combination of authentication methods would be user-to-token biometric authentication, followed by mutual cryptographic authentication between the token and system services. This combination may emerge sooner than one might imagine. Deployment of such technology on a large scale is certain to raise social and political debate.

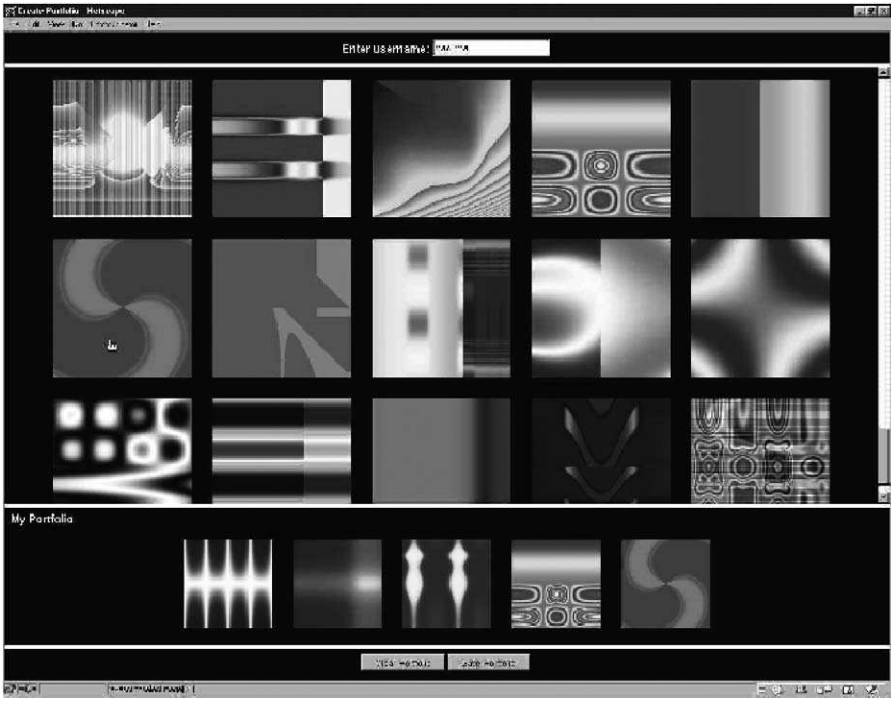


FIG. 7. Portfolio selection window [43].

2.3 Access Control

Access control, sometimes referred to as authorization in the literature, is the second guard in a secure DBMS. The prerequisite of an access control system is successful user authentication. All the discussion in this section is based on the assumption of legitimate users. The tasks of an access control system include defining access policy, determining and granting access rights for a specific user, and enforcing the execution of those rights.

When applied in different application domains, access control systems are expected to achieve different confidentiality and integrity objectives. Following similar ideas used in reusable software, it is necessary to distinguish the access control mechanism from the authorization policy. Policies are high-level guidelines that determine how accesses are controlled and access decisions determined [1]. It is usually the job of the database security administrator to define such policies. Mechanisms are low-level software and hardware functions that can be configured to implement a

TABLE VII
SUMMARY OF AUTHENTICATION METHODS

Method	Based on	Advantage	Disadvantage
Knowledge-Based	"Something that you know"	• Simple	• Password reusability • Passwords can be forgotten, shared, or observed
Token-Based	"Something that you possess"	• Simple • Overcomes some of weaknesses of knowledge-based systems	• Tokens can be forgotten, lost, stolen, or duplicated
Biometric-Based	Physiological or behavioral characteristic	• Greater security than other methods • Can distinguish between authorized person and imposters	• Privacy issues • Social acceptance
Recognition-Based	Recognition of previously viewed images	• Can overcome the drawbacks of knowledge- and token-based systems	• Most current proposals are not truly recognition based

policy [1]. The choice of different mechanisms is usually made by DBMS designers. Note that mechanisms should be flexible and comprehensive enough to enforce a variety of policies.

In this section we first introduce the basic principals in an access control system. Then we discuss the most common access control mechanism—the access matrix and its two implementation alternatives. Finally, we summarize three classes of authorization polices proposed in the literature: the discretionary, mandatory, and role-based access control policies.

2.3.1 Basic Principals

Subjects and objects are the two basic principals in an access control system. Subjects can be users or processes executing on behalf of users [1]. There is a many-to-many relationship between subjects and users: a user can log on to the system as different subjects and a subject can represent different users. Subjects are the entities that initiate activities in the system. Objects are the entities behind the protection of the system. In a DBMS environment, an object can be a field, a record, a table, a view, etc.

Subjects initiate actions or operations on objects. Operations that subjects can perform on objects (access rights) are usually defined by the security administrator and represented in the form of an access matrix, or one of its alternatives. For databases, the typical access rights include own, read, and write. Normally, the owner of a database is authorized to grant and revoke access rights to the database.

2.3.2 The Access Matrix

An access matrix is a common approach to modeling the access rights of subjects with respect to objects. Each row in the matrix represents a subject and a column corresponds to each object. Each entry in this matrix defines the access rights of a subject over an object. Figure 8 shows an access matrix example, where *O* represents ownership, *R* denotes the read right, and *W* represents the write right.

In this example, there are three subjects: Alice, Bob, and Cathy. There are four objects, namely Tables 1–4. Alice is the owner of Table 1 and Table 3, and therefore, she has the read and write rights to those tables. In addition, the matrix also specifies that Alice has the read right on Table 4, but she has not have access rights to Table 2.

When a user, say Bob, requests a read operation on Table 3, the request is sent to a program called the reference monitor. The reference monitor checks the access matrix to find out whether Bob has the read right to Table 3. In the example shown in Fig. 8, Bob does not have any access rights to Table 3. Thus, the Bob's read request will be rejected. Figure 9 illustrates the general authorization control flow.

It is not difficult to imagine that an access matrix representing a large system with many subjects and objects would be very sparse: A subject only has access to very few of the many objects. Storing access information in a matrix with many empty entries is obviously inefficient. Therefore, the access matrix is mainly used as a conceptual model, and one of its two alternatives is implemented in practical systems: the access control list and the capability list.

Object / Subject	Table 1	Table 2	Table 3	Table 4
Alice	O, R, W		O, R, W	R
Bob		O, R, W		R
Cathy	R			O, R, W

FIG. 8. An access matrix example.

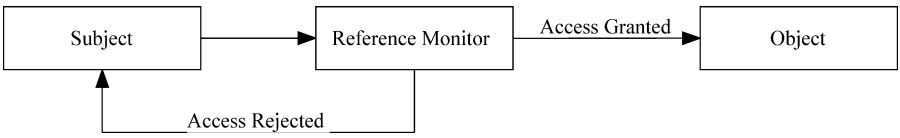


FIG. 9. General authorization control flow diagram.

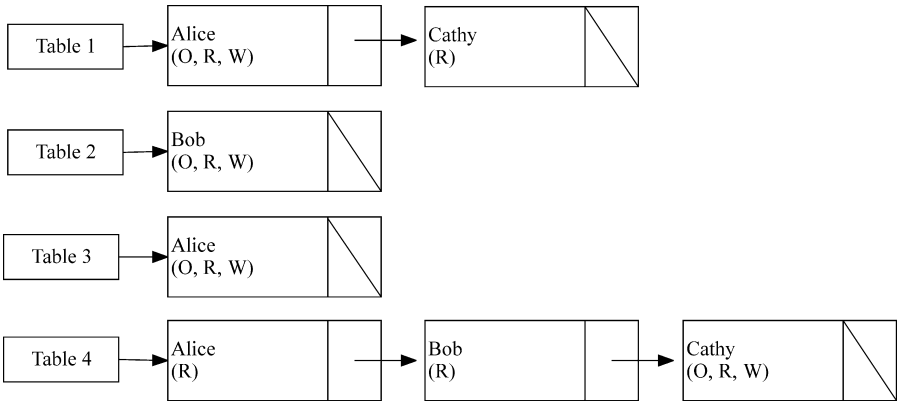


FIG. 10. The ACLs corresponding to the access matrix in Fig. 8.

2.3.2.1 Access Control List. In the Access Control List (ACL) approach, columns of the access matrix are stored eliminating the empty entries. Each object is associated with a list of subjects and the access rights to the objects each subject posses. Figure 10 depicts the ACLs corresponding to the access matrix shown in Fig. 8.

The major advantages of ACL-based approach are listed as follows:

- ACLs provide a convenient way to find out who can access a specific object with which privileges.
- Modifying the access rights associated with each object is easy.
- Revoking access to an object from all subjects can be done by simply removing the existing ACL.

On the other hand, determining the objects that a subject can access becomes very difficult. It requires a full scan of all ACLs. In practice, if a subject's access rights to all objects need to be revoked, the administrator may disable the subject's account rather than deleting the subject from each of the ACLs.

2.3.2.2 Capability List. Unlike ACLs, which store the access matrix by columns, a Capability List (CL) stores the matrix by rows. A CL provides information from a subject's point of view. It is essentially a list of objects that a subject can operate on. It also defines the access rights associated with each object that subject has. Figure 11 shows the CLs corresponding to the access matrix of Fig. 8.

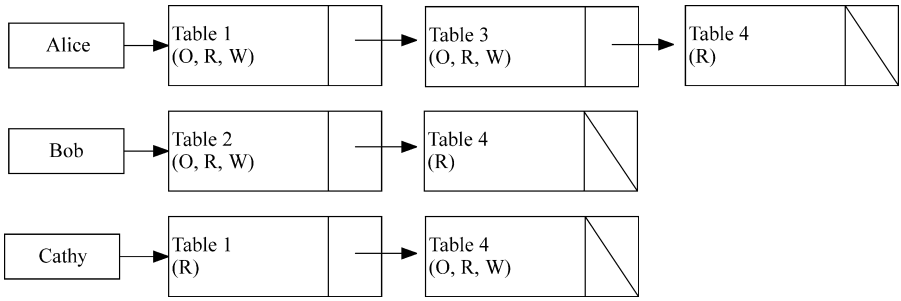


FIG. 11. The CLs corresponding to the access matrix in Fig. 8.

Sometimes the CL-based approach is called a dual approach to the ACL-based approach [1] indicating that the advantages and disadvantages of the two approaches are reversed. For instance, it is easy to modify or revoke the access of a subject to all objects in CL-base systems. However, modifying or revoking the access rights associated with a particular object requires examination of each CL in the system.

2.3.2.3 A Combinatory Approach. From the discussion in the previous sections, we have seen that the ACL-base and the CL-base approaches have their own strengths and deficiencies. One way to take advantage of both approaches is to represent the access matrix by an authorization relation [1]. In this relation, or table, each tuple consists of three fields: subject, access right, and object. Figure 12 shows the authorization relation of the access matrix shown in Fig. 8.

This representation is typical in relational DBMSs [1]. When sorted by subjects, the authorization relation is essentially a collection of capability lists. When the authorization relation is sorted by objects, we obtain the access control list of each objects.

2.3.3 Access Control Policies

The enforcement of access control policies relies on the underlying mechanisms introduced in Section 2.3.2. Generally speaking, there is no "best" policy. The appropriateness of a policy depends on the system security requirements. Three classes of policies are commonly used:

- Discretionary Access Control (DAC) Policies,
- Mandatory Access Control (MAC) Policies, and
- Role-Based Access Control (RBAC) Policies.

Subject	Access Right	Object
Alice	O	Table 1
Alice	R	Table 1
Alice	W	Table 1
Alice	O	Table 3
Alice	R	Table 3
Alice	W	Table 3
Alice	R	Table 4
Bob	O	Table 2
Bob	R	Table 2
Bob	W	Table 2
Bob	R	Table 4
Cathy	R	Table 1
Cathy	O	Table 4
Cathy	R	Table 4
Cathy	W	Table 4

FIG. 12. The authorization relation corresponding to the access matrix in Fig. 8.

It should be noted that these policies are not mutually exclusive. Instead, a combination of different policies may provide better system protection. When two or more policies are combined, rules defined by each policy must be satisfied. Sometimes, conflicts are inevitable: An operation considered to be legal in one policy may be defined as prohibited in another. An appropriate level of management should be consulted to resolve these discrepancies.

2.3.3.1 Discretionary Access Control Policies. Recall that an access matrix stores the access rights that a subject has to an object for all subjects and all objects in the system. Discretionary access control (DAC) policies use this information in a straightforward manner: They govern the subject's access to objects based on the subject's identity and authorization rules [1]. These rules can be represented in ACL or CL format. Figure 9 well illustrates the authorization process. Each subject's access request is checked against the authorization rules. If an existing rule states that the subject can access the object in the requested mode, the access is granted. Otherwise, it is rejected.

Discretionary access control policies can be further divided into two groups according to the choice of the default decision mode [45]: closed discretionary policies (positive authorization) and open discretionary policies (negative authorization). Closed discretionary policies assume a denial of access unless specified otherwise by the authorization rules. In contrast, open discretionary policies assume that an access request can be granted unless stated explicitly otherwise by the rules.

Flexibility is the major advantage of discretionary policies. However, they cannot control the flow of information in a system and therefore, it is easy to bypass the restrictions defined by the authorization rules. For example (see Fig. 8), Cathy has the read permission to Table 1, but Bob does not have any access to Table 1. A discretionary policy cannot prevent Cathy from reading the information of Table 1 and passing it to Bob. The cause of this problem is that discretionary policies do not impose restrictions on the flow of the information once it is obtained [1]. The class of access control policies that we will introduce next, mandatory access control policies, can address this problem and they are often used to augment the discretionary policies [46].

2.3.3.2 Mandatory Access Control Policies.

Mandatory access control (MAC) policies define authorization rules based on the classification of subjects and objects in the system [1]. Hierarchical classifications are made within the domains of confidentiality and integrity. System security is achieved by enforcing a set of read/write rules among the hierarchies. In this subsection, we first use two models to explain the essence of mandatory access controls with regard to information confidentiality and integrity: the Bell–LaPadula (BLP) model and the Biba Model. Then, we discuss a composite model that achieves both information confidentiality and integrity.

(1) *The BLP model*. Within the scope of information confidentiality (secrecy), each subject and object is assigned a security label [47]. A subject's security label, called security clearance, reflects the subject's trustworthiness not to disclose information to unauthorized subjects. An object's security label, called security classification, reflects the sensitivity of the object. The most common example of totally ordered security classes are the TS (top secret), S (secret), C (confidential), and U (unclassified) security levels used in the military and government systems [47]. In the order of TS, S, C, and U, the security or sensitivity level of a subject or an object decreases.

The BLP model is one of the earliest models that formalized the concept of mandatory access controls [47]. The motivation of the BLP model is to achieve information confidentiality. It combines the discretionary and mandatory access control methods. An operation can be carried out only when rules stated by both methods are satisfied. Let λ denote the security label of a subject or object. The mandatory access rules specified in the BLP model are as follows [47]:

- Simple-Security Property (read rule): Subject s can read object o only if $\lambda(s) \geqslant \lambda(o)$.
- *-Property (write rule): Subject s can write object o only if $\lambda(s) \leqslant \lambda(o)$.

In other words, a subject can only read objects of the same or lower security level, whereas a subject can only write objects of the same or higher security level. Enforcing these two rules guarantees that no information flows from a higher security level (more sensitive) to a lower one and therefore, ensure the confidentiality of information.

(2) *The Biba model*. Within the scope of information integrity, each subject and object is assigned an integrity label [47]. A subject's integrity level reflects the subject's trustworthiness for modifying information. An object's integrity level indicates the reliability of the information stored in that object, and the potential damage that could result from unauthorized modification. One example of the integrity hierarchy could be C (crucial), I (important), and U (unknown) [1].

The Biba model [48] is inspired by the idea of assuring information integrity. Biba proposed several ways to achieve information integrity. We will only introduce the most well known one, called strict integrity. Let ω denote the integrity label of a subject or object. The mandatory access rules specified in the Biba model are as follows [47]:

- Simple-Integrity Property (read rule): Subject s can read object o only if $\omega(s) \leqslant \omega(o)$.
- Integrity *-Property (write rule): Subject s can write object o only if $\omega(s) \geqslant \omega(o)$.

We notice that these properties permit a reversed direction of information flow from the direction regulated by the BLP model read/write properties. A subject can read objects of the same or higher integrity level and write objects of the same or lower integrity level. Satisfying these properties guarantees that no information of a lower integrity level (less reliable) flows to a higher one and therefore, ensures the integrity of information.

(3) *The composite model*. After the discussion of the BLP and Biba models, naturally one would think to build a model that combines the two so that both information confidentiality and integrity can be achieved. The composite model introduced in [47] is one such model. In this model, each subject and object is assigned two labels: a confidentiality label λ and an integrity label ω. The mandatory access rules can be stated as follows:

- Read Rule: Subject s can read object o only if $\lambda(s) \geqslant \lambda(o)$ and $\omega(s) \leqslant \omega(o)$.
- Write Rule: Subject s can write object o only if $\lambda(s) \leqslant \lambda(o)$ and $\omega(s) \geqslant \omega(o)$.

By guaranteeing these properties, we ensure that the system satisfies both information confidentiality and integrity.

2.3.3.3 Role-Based Policies. Role-based access control (RBAC) policies establish permissions based on the functional roles in the enterprise [49]. Roles can represent the tasks and responsibilities within an enterprise. With RBAC, a set of roles must be identified and users are assigned to an appropriate role or a set of roles. Instead of defining access rights to all objects for each user, the RBAC policies specify these rights on a role basis. A role acts as a mediator between a group of users and a set of tasks/responsibilities associated with them.

Roles in an enterprise are generally persistent and many users can be represented by a single role. Thus, RBAC approaches can reduce the administration complexity, cost, and potential errors. A number of RBAC models have been proposed and implemented [50–59]. A study by NIST (National Institute of Standards and Technology) shows that RBAC approaches are popular in commercial and government applications [60].

As an effort toward standardizing the RBAC approaches, NIST proposed a sequence of four RBAC models in increasing order of functional capabilities [49]:

- Flat RBAC Model,
- Hierarchical RBAC Model,
- Constrained RBAC Model, and
- Symmetric RBAC Model.

The requirements in these models are cumulative in the sense that the requirements in a more advanced model include all requirements defined in its predecessor. For instance, requirements in the hierarchical RBAC model embrace all requirements specified by the flat RBAC model. Moreover, additional conditions must be satisfied in the hierarchical RBAC model in order to achieve more complex functionalities than those provided by the flat RBAC. Table VIII summarizes the requirements for the four models in such cumulative fashion [49].

In [1], the authors summarize several advantages of role-based access controls as follows:

- *Simplified authorization management*: When a user's role changes in an organization, for instance due to a promotion, the system administrator can simply change the user to a new role. In contrast, if the access relations are defined between subjects and objects as in DAC-based approaches, all existing access rights must be identified and revoked, and new ones need to be assigned.
- *Hierarchical Roles*: The use of hierarchical roles captures the natural organization of an enterprise and is thus easy to understand and manage.

TABLE VIII
NIST RBAC MODELS

Model	Requirements
Flat RBAC Model	• Permissions (access rights) are assigned to roles. • Users acquire permissions by being members of roles. • The user-role and permission-role are many-to-many relation. • Roles assigned to a specific user can be determined. • Users belonging to a specific role can be determined. • A user can exercise multiple roles simultaneously.
Hierarchical RBAC Model	• A role hierarchy (a mathematical partial order defining the seniority relation between roles) is supported. • A role of higher seniority automatically inherits all permissions granted to lower seniority roles in the hierarchy.
Constrained RBAC Modes	• Separation of duties (SOD) must be enforced in order to reduce the possibility of fraud and accidental damage.
Symmetric RBAC Model	• Roles assigned to a specific permission can be determined. • Permissions belonging to a specific role can be determined.

- *Least Privilege*: Users can take roles that have the least privilege required to fulfill the current task. This minimizes the damages caused by unintentional errors.
- *Separation of duties* (*SOD*): The SOD method has been widely used in different real-world applications for reducing the possibility of fraud and inadvertent damages. Protection of the system is achieved by spreading responsibilities and authority for a task over multiple users, thereby raising the difficulty of committing a fraudulent act by requiring the involvement of more than one individual [49].

2.4 Auditing and Intrusion Detection

Auditing consists of examining the history of events in a system to determine if and how security violations have occurred or been attempted [1]. Auditing is required for all system activities: the subject requesting access, the object to be accessed, the operation requested, permissions granted or rejected, the resource consumption, the outcome of the request, etc. Audit data is stored in an audit log.

Depending on the system security requirements, audit data can be collected at different granularities. The finer grain audit data provide more detailed information and can assist in security violation detection. However, we must note that the volume of such data grows rapidly and will raise serious storage space concerns. Moreover, searching for violations among such massive amounts of data is difficult. Sophisticated intruders may spread their activities over a long period of time. As a result,

TABLE IX
THRESHOLD-BASED INTRUSION DETECTION APPROACHES

	Description	Characteristics
Predefined Threshold-Based Approach	• Based on the assumption that security violations usually involve abnormal use of the system. • Predefined allowable threshold values are used for intrusion detection.	• Cannot detect violations that do not overuse system resources. • Requires priori knowledge of system usage.
Anomaly-Based Approach	• Based on the concept of "normal" usage of the system. • Statistical models such as the operational model, the mean and standard deviation model, and time series model can be used to define normalcy. • Operations that fall out of the normal behavior range will trigger the security alert.	• Cannot detect violations that do not overuse system resources. • User behavior can be very erratic.

TABLE X
RULE-BASED INTRUSION DETECTION APPROACHES

	Description	Characteristics
Low-Level Rule-Based Approach	• Based on known intrusions and system vulnerabilities, rules are defined and used to describe suspicious activities. • Independent from users' behavioral patterns.	• Can only detect known attacks. • Requires expert knowledge to translate security threats into series of low-level system activities (audit records).
Model-Based Approach [61]	• Use models of proscribed intrusion activities for intrusion detection. • Models consist of scenarios, which are high-level observations of user behavior.	• Can only detect known attacks. • Security administrator describes high-level scenarios and leaves the translation from scenarios to audit records to the system.
State Transition-Based Approach [62]	• Models a violation as a sequence of actions starting from an initial state to a final compromised state. • A state represents the status of all volatile, semi-permanent, and permanent memory locations of a system at a particular time.	• Can only detect known attacks. • Only violations that change system states can be detected. • Like model-based approaches, high-level representation of violations is used, leaving the mapping from high to low-level description to the system. • A state definition can be used in describing different violations.

those violations are embedded in a huge amount of normal audit data. It is almost impossible to rely solely on the auditors to identify security threats accurately and quickly. Many automated tools have been developed to help human screening and analyzing audit information. These tools can be classified into different categories when viewed from various angles [1].

From the functionality point of view, automated tools can be classified as *information summarizing tools* and *intrusion detection systems*. The former category processes the raw audit data and generates summaries. The main purpose is to organize information and reduce the amount of information that human auditors need to review. These tools do not perform data analysis. The latter category can not only reduce the volume of data, but also analyze the data and detect intrusions.

Based on when the analysis is performed, intrusion detection systems can be further divided into *passive* and *active* systems. Passive systems perform a posteriori analysis and bring security violations to the auditor's attention. Active systems perform analysis in real time. Once a violation is detected or suspected, the intrusion detection system alerts the auditor and may take immediate measures for system protection.

From the intrusion detection method's perspective, intrusion detection systems may adopt the threshold-based and/or rule-based approaches. When examining these approaches in scrutiny, we can further identify the predefined threshold-based approach and the anomaly-based approach. They both belong to the threshold-based approach family but each has its unique characteristics. As variations of the rule-based approach, the low-level rule-based, model-based, and state transition-based approaches each have their pros and cons. Tables IX and X briefly summarize these approaches and highlight their prominent characteristics. None of the currently known approaches can provide a complete solution to intrusion detection problems. Thus, much research still needs to be done on this subject.

3. Security Issues of Multidatabase Systems

3.1 Multidatabase Systems

A multidatabase system is a collection of cooperating autonomous and heterogeneous databases. This structure allows one to share information and functionality to a precisely determined degree and allows more or less uniform access to the information sources [63]. Typically, such systems consist of two components: the global component and the local components. The global component is in charge of coordinating the local data sources (local components) such that they cooperate with each other and share common resources. Users interact with the global DBMS in a

TABLE XI
GLOBAL COMPONENT STRUCTURE

Centralized control	• A global administrator (GA) authenticates users, • GA authorizes user requests and routes them to the respective local databases, • GA integrates partial results and presents it to the user, • GA manages accountability of the aforementioned transactions, • GA restores the global system in a consistent state after accepting/removing new/existing databases.
Distributed control	• A local administrator (LA) is the administrator of the database system that joins a global system, • LA accepts requests directly from users without the intervention of the GA or global system (full autonomy), • LA accepts/rejects requests from the GA or global system, performs a security check and based on the result, returns a result back, • LA also performs a rollback if requested by the GA or global system.

uniform manner regardless of the heterogeneity of local DBMSs. The literature has addressed a wide range of solutions for global information sharing in a distributed environment. Different metrics can be used to distinguish these solutions from each other. A distinction can be made based on:

- The structure of the global component, or
- The type of interaction between the global component and the local components.

The structure of the global component can be centralized or distributed in nature. The properties of centralized and distributed global components are listed in Table XI.

The interaction between the global component and local components can be broadly defined as loosely coupled and tightly coupled. Table XII summarizes the major characteristics of systems based on this metric.

3.1.1 Loose Coupling

In a loosely coupled system, database can join or leave the system anytime at will. A local database decides which subset of its data will be contributed to the global system. The global component has the responsibility of searching for the location of relevant data with respect to a query. Every Component Database (CDBMS) acts like a server in this model. Such systems are flexible, but the global system is burdened with many system management responsibilities.

Based on the coupling relation between the global and local components, an interoperable system is at the extreme end of the spectrum for a collection of database ob-

TABLE XII
INTERACTION BETWEEN LOCAL/GLOBAL COMPONENTS

Loosely coupled	• Global schema is loosely defined. It has tangible access to low level, internal schemas of local databases. • Global access to local data is through interfaces provided by every local database. • Full autonomy is easily achievable for loosely coupled systems. • Highly dynamic—local databases can easily join and leave the system. • Synchronization is possible through mechanism such as message passing or remote procedure call.
Tightly coupled	• Global schema has access to low level, internal schema of local databases. • Allows close synchronization. • More static in nature. • Full autonomy at local databases is hard to achieve. • Efficient global query processing.

jects. Dynamic interfaces or common standard APIs need to be developed to process knowledge structures (heterogeneous databases) in order to transform data to useful information. This dynamic interface must have the following functions [64]:

- Methods to transform and sub-set databases using view definitions and object templates;
- Methods to access and fuse data from multiple databases;
- Computations to support abstraction and generalization over underlying data;
- Intelligent directories to information bases, such as library catalogs, indexing aids, and thesaurus structures; and
- Methods to deal with uncertainty and missing data due to incomplete or missmatched sources.

Defining wrappers and mediators in heterogeneous environments provides a feasible implementation of the aforementioned features. These approaches are based on mediation and do not require the specification and management of global schema or common language for communication.

Data sharing does not mandate sharing of representation. As long as heterogeneous databases can communicate with each other, they can benefit from each other's data without being bound by the problem of common representation. Centralized databases mandate physical database sharing, while distributed databases mandate logical but not physical database sharing. Federated/multidatabases mandate schema but not database sharing. This sharing process opens the local system to vulnerabilities and enlarges the space for possible threats to the data and resources it maintains. The major requirements of integration/mediation efforts with respect to security issues are as follows:

- *Transparent access*: Application users should be able to access information from any or all of the sources in a uniform and consistent way. Specialized knowledge of individual sources should not be required.
- *Autonomy*: Application users must be permitted access to information they can access from any individual source. Furthermore, the sources must continue to function after integration as they did prior to integration.
- *Security*: Users of applications must be denied access to any information that they cannot access from any source individually.

3.1.2 Tight Coupling

Tightly coupled systems provide global services, offered by a global schema, running on top of the CDBMS. Users submit requests at a single interface. An administrator manages this tightly coupled system. The administrator also imposes any security mechanisms on user requests. These mechanisms typically include authentication, authorization, and access control. Users, however, also have the liberty to request data at any CDBMS. The local administrator in this case needs to provide the required security mechanisms. Global users in tightly coupled systems are abstracted from the location of data (location transparency) and resolution of semantic heterogeneity. The global security administrator (SA) negotiates with the local SAs about the security policies to be enforced and sets up a global access control system. A distributed database system is the most tightly coupled information sharing system [65]. Global and local components share very low level and internal interfaces—there is little distinction between them. In general, there are two approaches to the design of a distributed database management system (DDBMS). One approach involves combining existing database management systems [66]. However, DDBMS are typically designed from scratch in a top down fashion. When a DDBMS is designed from scratch, the local DBMS are typically homogeneous and the global system has control over local data and processing. The system maintains a global schema by integrating the local schemas. Global users access the system by submitting queries at the global system. Distributed databases have the best performance at the cost of significant local modification and loss of control.

The coupling degree of the global and local components can be extended, classifying global information-sharing solutions into six main categories [65] as shown in Table XIII. Of these solutions, the federated database system is of most interest to researchers. Sometimes, the terminology multidatabase and federated database are used interchangeably. Therefore, we will use the federated database as the default underlying architecture when discussing the security issues in multidatabase systems (MDBMSs). The term federated database system was coined by Hammer and McLeod [67] and Heimbigner and McLeod [68]. A federated database system

TABLE XIII
GLOBAL INFORMATION SHARING ENVIRONMENT [65]

Types	Features
Distributed databases	• The most tightly coupled information sharing system
Global schema multidatabases	• A loosely coupled flavor of distributed databases • The local databases are heterogeneous in nature • A user submits a query at the global schema level • Information about data in a particular database need not be maintained
Federated databases	• A more loosely coupled flavor of global schema multidatabases.
Multidatabase Language systems	• More loosely coupled than global schema multidatabases or federated databases • No global schema is maintained • Information in local databases must be known by users
Homogenous multidatabase language systems	• A degenerate form of multidatabase language systems • The participating local databases are homogeneous in nature
Interoperable systems	• The extreme flavor of loose coupling in a collection of database objects • The local databases can be knowledge-based information systems, or expert systems

(FDBS) is a collection of cooperating but autonomous component database systems (CDBSs). The CDBSs are integrated to various degrees. The software that provides controlled and coordinated manipulation of the component DBSs is called a federated database management system (FDBMS). The DBMS of a component DBS can be a centralized or distributed DBMS or another FDBMS.

FDBSs can be characterized along three orthogonal dimensions: distribution, heterogeneity, and autonomy [69]. Data may be distributed among multiple databases. These databases may be stored on a single computer system or multiple computer systems, co-located or geographically distributed but interconnected by a communication system. The data may also be distributed in different ways. In relational terms, these methods include vertical and horizontal database partitions. Multiple copies of some or all of the data may be maintained. These copies need not be identically structured. Differences in DBMSs include differences in structure, constraints, and query languages. These variations lead to the heterogeneity of FDBS. Semantic heterogeneity occurs when there is a disagreement about the meaning, interpretation, or intended use of the same or related data. Veijalainen and Popescu-Zeletin [70] classified the autonomies into three types: design, communication, and execution.

Autonomy is the ability of a database system to perform operations independently. Local autonomy means that the local administrator maintains control over local resources. The requirement for autonomy is two-fold. First, the local components described earlier need to "share data" to cooperate and form a coalition. Secondly, these components also need to maintain authority over local resources to forbid any policy violations. Table XIV classifies the different types of autonomy and gives a brief explanation of each.

Association Autonomy can be further classified as shown in Table XV.

Design autonomy refers to the ability of a CDBS to choose its own design with respect to any matter. Examples of such design considerations include data management, the representation and the naming of the data elements, the conceptualization or semantic interpretation of the data, constraints used to manage the data, the functionality of the system, the association and sharing with other systems, and the implementation. Communication autonomy refers to the ability of a component DBS to decide whether to communicate with other component DBSs. A component DBS with communication autonomy is able to decide when and how it responds to a request from another component DBS. Execution autonomy refers to the ability of a component DBS to execute local operations without interference from external operations and to decide the order in which to execute external operations. Association

TABLE XIV
AUTONOMY CLASSIFICATION AND EXPLANATION [71]

Classification	Description
Design autonomy	The ability of a component system to choose the organization and accessibility of the information it stores with reference to the data model and query language to be used.
Communication autonomy	The ability of a component system to decide whether to communicate with (i.e., to respond to the requests of) other component systems.
Execution autonomy	The ability of a component system to execute local operations without interference from external operations.
Association autonomy	The ability of a component system to decide whether and how to share its functionalities (operations and resources) with other systems.

TABLE XV
ASSOCIATION AUTONOMY CLASSIFICATION [71]

Authentication	*Authentication autonomy* is the ability of a system to establish an identity of each user accessing the system.
Authorization	*Authorization autonomy* is the ability of a system to specify which accesses are to be allowed or denied on objects stored at the system.

autonomy implies that a component DBS has the ability to decide whether and how much to share its functionality and resources with others. This includes the ability to associate or disassociate itself from the federation and the ability of a component DBS to participate in one or more federations. Association autonomy may be treated as part of the design autonomy or as a type of autonomy in its own right. Alonso and Barbara [72] discuss the issues that are relevant to this type of autonomy.

As can be concluded from Table XIII, multidatabase systems can be elaborated as co-operations of autonomous and heterogeneous databases. Multidatabase systems integrate data from pre-existing, heterogeneous local databases in a distributed environment and present global users with transparent methods to uniformly access and manipulate the total information in the system [65,73,74]. A multidatabase system, generically, can be further classified as: global-schema multidatabase, federated databases, multidatabase language system, and homogeneous multidatabase language systems [74]. The multidatabase differs from an inter-operable system because it provides full database functionality to global users. It differs from a distributed database in that the local databases in a multidatabase system preserve full autonomy.

The literature has introduced two general approaches to resolve multidatabase heterogeneity, namely, the global schema approach and multidatabase languages. The first approach is inherited from traditional distributed databases where local schemas are integrated into a global schema representing common data semantics. This approach provides a high degree of transparency. However, the schema integration process is rather complicated and labor intensive. The meta-data, global schema, is usually maintained at every site to allow simple and fast accesses to the data. Duplication of the global schema, however, raises the consistency problem in the case of updates at local databases.

In the second approach, the integration of local schemas is achieved through a common multidatabase language that interprets and transforms a query to data represented and maintained at local databases. There is no global view of shared data. Multidatabase languages, however, provide many useful features to allow the global user to formulate a request. Nevertheless, a user must know the location and access terms of the data being queried. The choice between the global schema paradigm and the multidatabase language approach is a trade off between efficiency and transparency.

The *Summary Schemas Model* (SSM) [65] is a compromise between the aforementioned two approaches; an adjunct to multidatabase language systems that supports automatic identification of semantically similar/dissimilar data entities. The model maintains a hierarchical meta-data based on access terms exported from underlying local databases. This meta-data is used to resolve name differences using word relationships defined in a standard dictionary such as Roget's Thesaurus.

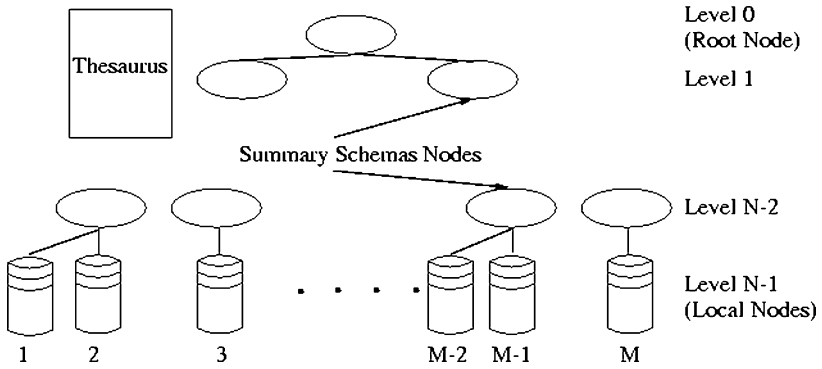


FIG. 13. A Summary Schemas Model with M local nodes and N levels.

As depicted in Fig. 13, the SSM consists of three major components: a Thesaurus, the local nodes, and the summary-schemas nodes. The thesaurus defines a set of access terms, namely global terms, the semantic categories they belong to, and their semantic relationships. The thesaurus uses a Semantic-Distance Metric (SDM) to provide a quantitative measurement of "semantic similarity" between terms [65].

A local node is a physical database containing real data. A summary-schemas node is a logical database that contains a meta-data called *summary schema*, which represents the concise and abstract contents of its children's schemas. Fewer terms are used to describe the information in a summary schema than the union of the terms in the input schemas while capturing the semantic information of the input terms, and therefore, reducing the schema size at higher SSM levels. Relative to other multidatabase solutions, the SSM is a robust approach that preserves local autonomy, provides high performance, and has good scalability.

Research issues in multidatabase systems have been well addressed and documented in the literature [65,66,71,75,76]: autonomy, data representations, heterogeneity, global query processing, global query optimizations, concurrency control, and consistency. Security, however, has not received enough attention [66,71,73,75]. Security enforcement must take into consideration the protection requirements and the protection policies of each participating site [71]. The task of enforcing security is complicated due to the heterogeneity of the participating systems. Moreover, the integration of data from different sources and replication of the data at the global level for distribution to users further complicates security enforcement. Integrated administrative policies need to be sought as agreement of all individual policies.

Table XVI lists some of the security issues that must be considered in multidatabase systems. Security of information in multidatabase systems is even more important than in centralized and traditional distributed database systems because the

TABLE XVI
SECURITY ISSUES AND POSSIBLE SOLUTIONS IN MULTIDATABASES

Issues		Solutions
Authentication		Authentication is a process of proving that a user possesses the asserted identity. It is the necessary prerequisite for access control.
	At Global schema level	• Authenticate users to system only after verification. • Authenticate users to system without verification. (This is generally seen in loosely coupled databases.)
	At Local database level	• Authenticates users only after verification of the identity of the user (autonomous system). • Automatically authenticate users to the local system if the global schema accepts the user. (Delegates the authentication to the multidatabase global schema.)
Access Control		• Access control defines mechanisms required to: • Verify whether or not a request issued by a subject is allowed, and • Enforce the corresponding decision [75]. • Access control is a necessary prerequisite for Authorizations.
	At Global schema level	• Local identity to issue access to user (assigned by local database). • Unique global identity to decide access to user (assigned by multidatabase global schema). • Remote identity to allow access to local component (assigned by third party like trusted key distribution center and agreeable to both multidatabase and local component).
	At Local database level	• Local identity to issue access to the local component. • Unique global identity to decide access to user. • Remote identity to allow access to local component.
Inferential Security		Security when the user may use logical reasoning to infer a supposedly restricted piece of information.
Integrity		After integration of component databases in a community of databases, one might specify additional global integrity constraints. Any further modifications at local databases should not violate these integrity rules.

scale of databases is larger and the number of users involved is greater. However, the characteristics of multidatabase raise new challenges to this task. In the following subsections we will discuss the authentication, access control, and auditing in multidatabase systems, more specifically, FDBSs.

3.2 Multidatabase Authentication Mechanisms

In Section 2.2 we discussed authentication issues and solutions in centralized DBMSs. In the context of multidatabase systems (MDBMSs), the goal of authen-

tication is the same as it is in centralized DBMSs—identifying one party to another. However, the problem of authentication is far more complex in MDBMSs due to the distribution of the parties that are involved.

Users, workstations, communication channels, and services are the basic components of a distributed system, while users and workstations are the major principals in a centralized system. We can view the authentication problem in MDBMSs as an extension of authentication in centralized DBMSs. In a centralized environment, services are provided by local workstations. Thus, the authentication is between users and workstations (Fig. 14). In a distributed system, such as MDBMS, users often log on to local workstations in order to access services provided by remote workstations. The user must be authenticated at a local workstation first. Then the local workstation, acting on the user's behalf, is mutually authenticated with the remote service provider (Fig. 15). Because communication links are involved in the authentication process, countermeasures must be taken to handle eavesdropping, replay attacks, and masquerading. Without secure communication channels, the authentication system can be easily compromised. In the rest of the discussion, we assume that the local authentication is taken care of. Thus, the local workstation can interact with the rest of system on behalf of the authenticated user. We use the term "user" and "local workstation" interchangeably. Moreover, we follow the same notation summarized in Table I.

Researchers have proposed many authentication protocols for distributed systems. Based on these techniques, we can classify the protocols as symmetric cryptosystem based challenge-response authentication, asymmetric cryptosystem based challenge-response authentication, router-based authentication, and agent & model based au-

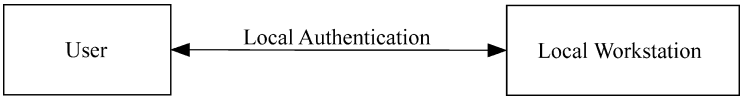


FIG. 14. Authentication in centralized DBMS.

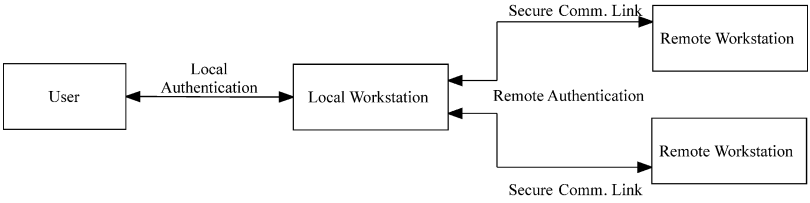


FIG. 15. Authentication in distributed DBMS.

thentication. They can be evaluated based on the following criteria [77]: effectiveness, granularity, flexibility, and performance.

3.2.1 Symmetric Cryptosystem Based Challenge-Response Authentication

3.2.1.1 Basic Protocol. Symmetric cryptosystem based challenge-response authentication protocols are designed according to the following principle called SYM [78], "*If a principal can correctly encrypt a message using a key that the verifier believes is known only to a principal with the claimed identity* (*outside of the verifier*)*, this act constitutes sufficient proof of identity*."

The success of these protocols relies on two basic assumptions:

- The underlying cryptosystem is strong—one cannot create the encrypted version of a message and cannot decrypt an encrypted message without knowing the secret key.
- Only the two parties involved in the authentication and no one else know the shared secret key.

Assuming that a user **A** sends an authentication request to a service provider **S**, Fig. 16 depicts the basic symmetric cryptosystem based challenge-response protocol that mutually authenticates **A** and **S** [79]. **A** first identifies himself/herself by sending the user ID "A" to the service provider **S**. **S** looks up its user-key list and finds the

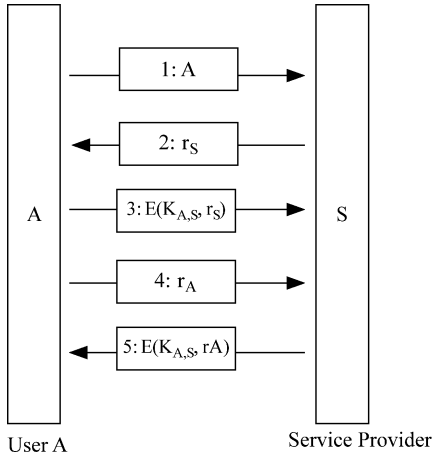


FIG. 16. Secret key based mutual authentication [79].

shared key between **A** and itself, $K_{A,S}$. **S** sends a challenge (a randomly generated number) $\mathbf{r_S}$ to **A**. **A** encrypts r_S with the shared key $K_{A,S}$ and sends it to **S** as the response. **S** decrypts **A**'s response with the shared key to authenticate **A**. By reversing the role of **A** and **S**, **A** can authenticate **S** in a similar way—**A** sends a challenge to **S** and **S** must respond with the correctly encrypted challenge.

This mechanism has several known disadvantages:

- It has poor scalability because every principal in the system must store the secret keys for all other principals that it may need to authenticate.
- Secret key distribution is a problem.
- The compromise of one principal could compromise the entire system.

These problems can be significantly alleviated by deploying a centralized authentication server that maintains a secret key shared with every principal in the system [80]. This authentication server is often implemented as a Key Distribution Center (KDC) in real systems. The well-known Needham–Schroeder authentication protocol exemplifies the use of a KDC [80]. Figure 17 shows a variation of the original protocol [79].

The main idea of this protocol is that if **A** wants to request services from **S**, he/she first needs to acquire a service ticket $E(K_{S,KDC}, (A, K_{A,S}, r_{S1}))$ from the KDC. This ticket contains the identity of the service requestor (in this case **A**), a secret key $K_{A,S}$ that will be used by **A** and **S** in the subsequent conversations, and a nonce r_{S1} to help

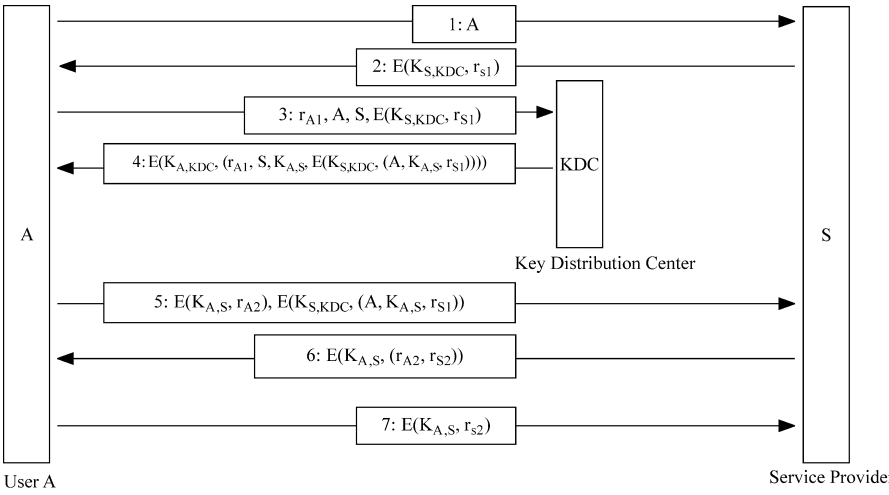


FIG. 17. A variation of the Needham–Schroeder authentication protocol [79].

preventing replay attacks. The ticket is encrypted by the shared key of the service provider **S** and the KDC. Thus, after **A** obtains the ticket, he/she cannot interpret the content. From the reply of KDC, **A** also gets a copy of the secret key $K_{A,S}$. The interaction between **A** and **S** after **A** receives the KDC's reply is similar to the scenario in Fig. 16. Therefore, we do not reiterate the rest of the protocol.

There are two noteworthy points in the protocol presented in Fig. 17:

(1) The use of random numbers (called nonce) r_{S1}, r_{S2}, r_{A1}, and r_{A2} is essential to prevent replay attacks. The purpose of a nonce is to uniquely identify a message and thereby, replayed messages can be immediately recognized. As a counter example, let's assume the following:

- Nonces r_{A1}, and r_{A2} are not in use.
- An intruder **T** has stolen the old $K_{S,KDC}$ and it also intercepted the old reply $E(K_{A,KDC}, (r_{A1}, S, K_{A,S}, E(K_{S,KDC}, (A, K_{A,S}, r_{S1}))))$ sent by the KDC.
- Once **S** notices that the old key $K_{S,KDC}$ is compromised, it immediately renegotiates with the KDC to set up a new $K_{S,KDC}$.

Since the reply from the KDC to **A** is intercepted by **T**, **A** will try to reissue a request to the KDC. At this point, **T** can intercept the request and replay the old reply. **A** cannot tell whether the reply received is for an old request or the latest one. Because **T** has the old $K_{S,KDC}$, it can masquerade as the service provider in the rest of the conversation. On the contrary, if a nonce is associated with **A**'s request to the KDC, **A** will notice right away when **T** replays the old KDC's reply because **A**'s new request has a different nonce than the one in the old reply.

Follow the same reasoning, the nonce r_{S1} used in message 2 is intent to prevent a intruder **T** who knows an old shared key $K_{A,S}$ from replaying message 5.

(2) The information "S" in message 2 is important to prevent man-in-the-middle attacks. Let us use a counter example again. Assume the following:

- "S" is not included in message 2.
- An intruder **T** can intercept all messages sent by **A**.

T can masquerade **S** by doing the following:

- Intercept message 1 and replace "S" with "T." By doing so, **T** makes the KDC believe that **A** wants to request services from **T** instead of **S**. Thus, the KDC will generate a shared key $K_{A,T}$ for the two parties.
- When **A** receives the reply from the KDC, **A** cannot tell the shared key is with **T** but not **S**.

- **T** can intercept all following messages that **A** sends to **S**. Since **T** has all the information it needs ($K_{T,KDC}$ and $K_{A,T}$) to decrypt the messages, it can act on **S**'s behalf without being noticed by **A**.

3.2.1.2 An Example—The Kerberos Authentication Service.

Kerberos is a distributed authentication scheme developed as part of Project Athena, at the Massachusetts Institute of Technology (MIT) [81]. It uses a variation of the Needham–Schroeder authentication protocol. Here we mainly focus on the authentication protocol, not the system itself. Interested readers can find more comprehensive information on Kerberos in [81–86].

Five principals are involved in the authentication protocol: users, local workstations, service providers, the centralized Key Distribution Center (KDC), and the centralized Ticket Granting Service (TGS). Figures 18–20 illustrate the message exchanges among the principals during an authentication [2]. Before we elaborate the procedure, it would be helpful to first highlight several assumptions and notations used in these figures:

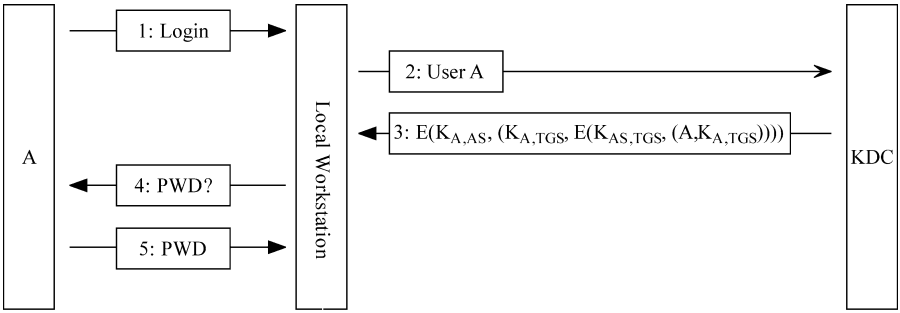


FIG. 18. The user-KDC interaction phase.

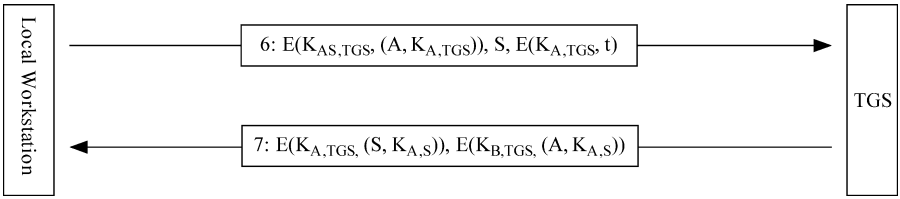


FIG. 19. The user-TGS interaction phase.

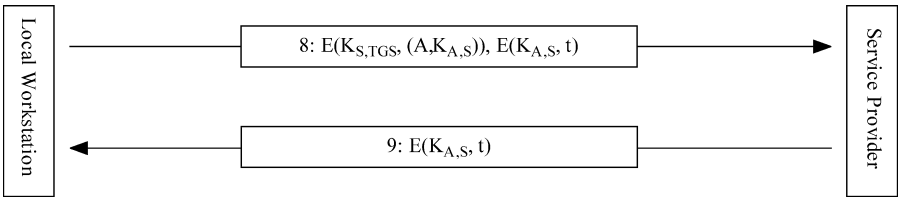


FIG. 20. The user-service provider interaction phase.

- User A logs on and is authenticated by the distributed system through his/her local workstation.
- User passwords are never stored on any of the components of the system.
- The secret key shared by a user and the KDC can be generated by an algorithm residing on the local workstations. This algorithm takes a user's password as the parameter for the secret key generation.
- The KDC maintains a shared key for each user in the system. In other words, the shared secret key of A and the KDC ($K_{A,KDC}$) is stored in the KDC.
- Key $K_{KDC,TGS}$ is the secret key shared by the KDC and the TGS.
- Key $K_{A,TGS}$ is a session key generated by the KDC at the time of A's log in request.
- Key $K_{A,S}$ is a session key generated by the TGS at the time of A's service request.

The authentication process can be divided into three phases: the user-KDC interaction phase, the user-TGS interaction phase, and the user-service provider interaction phase.

(1) *The user-KDC interaction phase*
   - User **A** types in a login request at a local workstation by providing his/her user name. The workstation sends **A**'s username in plain text to the **KDC** (message 2).
   - The **KDC** retrieves the secret key it shares with user **A**, $K_{A,KDC}$, from its local data store, generates a session key for **A** and the **TGS** ($K_{A,TGS}$), composes a ticket that **A** can use to request service from the **TGS** ($E(K_{KDC,TGS}, (A, K_{A,TGS}))$), and encrypts the session key and the ticket using the key $K_{A,KDC}$ (message 3). This message is sent back to **A**'s local workstation.
   - The local workstation requests **A**'s password. This password is used to generate the secrete key $K_{A,KDC}$. Message 3 is decrypted using the generated

key. The ticket for **A** to hand to the **TGS** and the secret key shared by **A** and the **TGS** are extracted from the deciphered message.

(2) *The user-TGS interaction phase*

- User **A**'s workstation packs the ticket for the **TGS**, the service provider's information **S**, and a timestamp t enciphered by the session key of **A** and **TGS** into message 6. This message is sent to the **TGS**. The **TGS** uses the timestamp t as a guidance to determine whether or not to issue a ticket for **A**, and thereby prevents replay attacks. If t is much smaller than the current time, the **TGS** will reject **A**'s request.
- Using its secret key $K_{\mathrm{KDC,TGS}}$, the **TGS** first decrypts the ticket to obtain the session key $K_{\mathrm{A,TGS}}$. Then, it generates a session key for **A** and its service provider **S**, $K_{\mathrm{A,S}}$.
- The **TGS** composes message 7 which includes a ticket for **A** to hand over to the service provider **S** and the session key of **A** and **S** encrypted by the session key of **A** and **TGS**. Message 7 is then sent to **A**'s workstation.
- After deciphering message 7, **A** obtains the ticket needed to requesting **S**'s service and the session key for **A** and **S** subsequent conversation.

(3) *The user-service provider interaction phase*

- **A**'s workstation composes message 8 which includes a ticket for **S**, and a timestamp encrypted by the session key of **A** and **S**.
- After **S** decrypts message 8 and replies message 9, **A** and **S** finish the mutual authentication process.

One important advantage that the Kerberos system provides is that user passwords are never stored on any machines or transferred through the network in plaintext. This significantly reduces the chance of stealing passwords from physical data stores and eavesdropping over the network by attackers. Nevertheless, Kerberos exhibits the following vulnerabilities [77]:

- The KDC must maintain a secret key for each principal in the system. This creates a system bottleneck and imposes high requirements on the server.
- The strength of Kerberos relies on strong crypto algorithms.
- Kerberos is vulnerable to Trojan horse software.

3.2.2 Asymmetric Cryptosystem Based Challenge-Response Authentication

3.2.2.1 Basic Protocol. In an asymmetric (ASYM) cryptosystem, each principal A possesses a pair of keys: the public key (K_{A}^{+}) and the private key (K_{A}^{-}). The public key is known to the public and the private key is kept secret by the prin-

cipal. Messages encrypted by a private key can be decrypted by the corresponding public key, and vice versa. The ASYM design principle [78] is as follows: "*If a principal can correctly sign a message using the private key of the claimed identity, that act constitutes a sufficient proof of identity*."

In Section 3.2.1 we mentioned that the existence of a centralized online key distribution center (KDC) that maintains the secret keys of all principals in a symmetric cryptosystem raises storage and single point of failure problems. In an asymmetric cryptosystem, there is no such central online key storage; instead, the key distribution occurs offline. A well-known entity called the certification authority (CA) is in charge of issuing the public-key certificate (a public-key certificate consists of a public key and the entity's ID to which the key is associated) when a principal registers to the system. To prevent forgery, all public-key certificates are signed by the CA's private key. Since the CA is well known, we assume that its public key is also well known. The certificates are kept by the principals and are forwarded to other principals during an authentication exchange. By comparing the (*public key, user ID*) pair provided by an entity with the information recovered from the certificate of that entity, a principal can be sure that the provided public key does belong to the entity that claims it [79].

Figure 21 shows a simple protocol (notations follow Table I) that utilizes an asymmetric cryptosystem [79]. This protocol is based on the assumption that user **A** and the service provider **S** know each other's public key. User **A** first challenges **S** with a random number r_A. This challenge and **A**'s user ID are encrypted by **S**'s public key K_S^+ (message 1). After decrypting message 1, the service provider generates a session key $K_{A,S}$ and a challenge r_B for **A**, and then composes message 2. To assure that only **A** can read this message, message 2 is encrypted by **A**'s public key K_A^+. **A** extracts the session key and the challenge from message 2 and encrypts r_B using the session key. Once **S** receives message 3, mutual authentication is achieved.

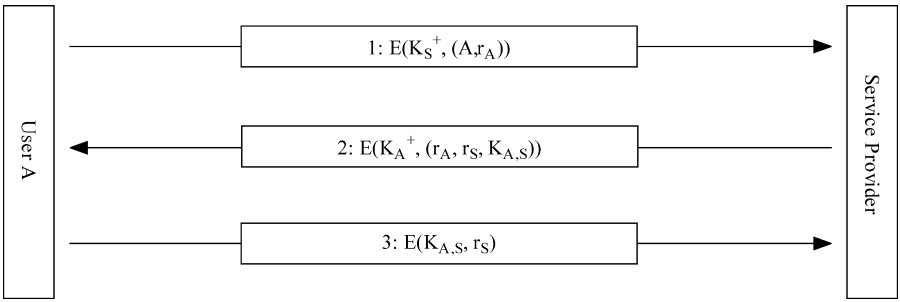


FIG. 21. Mutual authentication using an asymmetric cryptosystem.

3.2.2.2 An Example—The SPX Authentication Service. SPX is an authentication service using public-key certificates [87]. It is intended for open network environments. It is a major component of the Digital Distributed System Security Architecture [88]. Five principals can be identified in the SPX system [87]:

- Users are those who requests services.
- Service providers provide services to legitimate users.
- Certification Authorities (CAs) are entities that operate offline and are selectively trusted by principals. Their responsibility is to issue public-key certificates to users.
- The Login Enrollment Agent Facility (LEAF) is used in the credential initialization process. It role corresponds to the KDC in the Kerberos.
- The Certificate Distribution Center (CDC) resembles the TGS in Kerberos. It is an online depository of public-key certificates for all users, service providers, and CAs. It also stores encrypted private keys of users and service providers.

Assuming the scenario that user **A** requests services from service provider **S**, the credential initialization phase and subsequent authentication phase are described in Figs. 22 and 23, respectively [78]. Our purpose of presenting the SPX is to show a real-world application of the asymmetric cryptosystem based authentication mechanism. Thus, we focus on the high level description of the authentication protocol used by the SPX. For more details about the SPX system, readers can refer to [87]. In these figures, we follow the notation in Table I. In addition, the following notations are used:

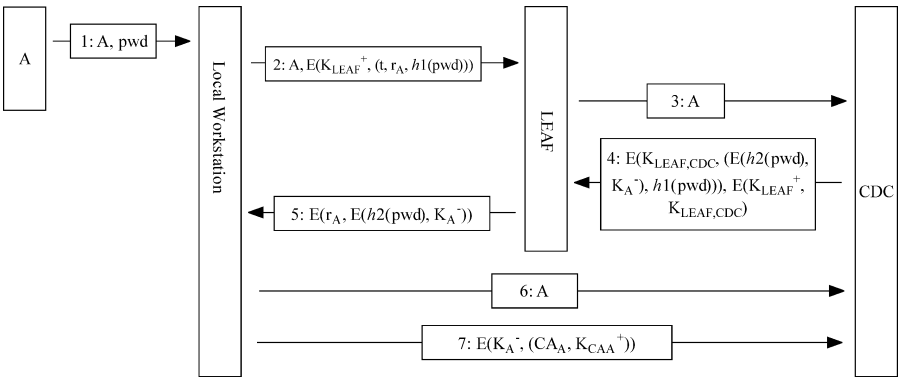


FIG. 22. SPX credential initialization phase.

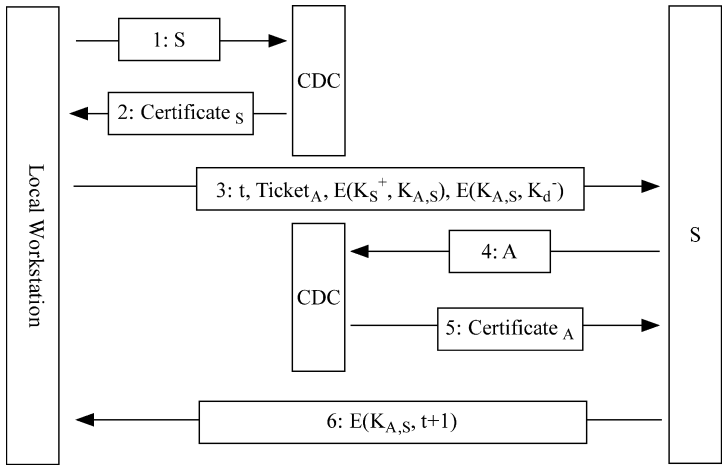


FIG. 23. SPX authentication phase.

- "pwd" is user **A**'s password.
- $h1$ and $h2$ are two publicly known one-way hash functions. "$h1(x)$" means applying hash function $h1$ on x.
- L represents the lifetime of a ticket.
- $\mathrm{CA_A}$ represents the certificate authority trusted by user **A**.
- $(K_{\mathrm{d}}^{+}, K_{\mathrm{d}}^{-})$ is a pair of public-private key (delegation key) generated by the RSA algorithm.
- $\mathrm{Ticket_A}$ is a ticket user **A** needs to hand over to **S**. $\mathrm{Ticket_A} = E(K_{\mathrm{A}}^{-}, (L, \mathrm{A}, K_{\mathrm{d}}^{+}))$.
- $\mathrm{Certificate_A}$ is the public-key certificate of **A**. It includes **A**'s public key and **A**'s ID. The certificate is signed by one of **A**'s trusted certificate authorities (K_{CAA}^{-}). $\mathrm{Certificate_A} = E(K_{\mathrm{CAA}}^{-}, (\mathrm{A}, K_{\mathrm{A}}^{+}))$.

In the credential initialization phase, user **A** logs on to a local workstation by providing a user ID and password. **A**'s ID and the encrypted password are sent to the **LEAF** in message 2. **LEAF** then contacts the **CDC** for **A**'s private key (K_{A}^{-}) information. The **CDC** establishes a session key with the **LEAF** and sends the encrypted **A**'s private key back to the **LEAF**. The **LEAF** forwards (message 5) the encrypted K_{A}^{-} to the local workstation. The local workstation recovers **A**'s private key from message 5. It then generates a pair of delegation key $(K_{\mathrm{d}}^{+}, K_{\mathrm{d}}^{-})$ using the RSA al-

gorithm, and creates a ticket to be used when acquiring services. Finally, the local workstation acquires the public key of a **CA** that user **A** trusts from the **CDC**.

During the authentication phase, user **A**'s workstation uses all the information obtained from the initialization phase to mutually authenticate a service provider. First, it requests the public-key certificate of **S** from the **CDC**. It then generates a session key $K_{A,S}$ and encrypts K_d^- using it. Message 3, containing A's ticket, the session key, the delegation key, and a timestamp, is sent to **S**. When **S** receives **A**'s service requests, it asks the CDC for **A**'s public-key certificate. **S** recovers K_d^+ and K_d^- from message 3. If it is verified that K_d^+ and K_d^- is from a delegation key pair, **S** believes that **A** is authenticated. To identify itself to **A**, **S** responds to **A**'s challenge in message 6. At this point, the mutual authentication is achieved.

3.2.3 Router Based Authentication

Router based authentication represents a simple form of an authentication mechanism. It is much less common than the previously introduced authentication mechanisms. The idea behind this technique is to enforce distributed authentication and access control through routers by applying routing constraints [77]. In practice, this is accomplished at three levels:

- At local workstations, users are authenticated by traditional methods such as password authentication.
- Access to LAN-based resources is controlled by the Network Information System (NIS).
- WAN routers provide protection to different domains based on a requesting workstation's address.

Routers recognize packages by senders' network addresses instead of users' identities. This method is only useful in simple environments, because it is difficult to achieve fine-grain security control.

3.2.4 Agent & Model Based Authentication

Public key cryptography (asymmetric cryptosystem), though very powerful in concept, raises a crucial issue—the problem of public key distribution. Certificates in such systems actually acknowledge the relation between the name and the object's public key. Unfortunately, many people have the same name. Therefore, only the certificate authority knows how to associate a name with a key. Many certification problems stem from misunderstanding the concept of identity and the phenomenon of trust. Hurton investigated the problem of identification and authentication in distributed systems from a different perspective and proposed the Agent & Model based

authentication mechanism in order to cope with the identity-key association problem [89]. The term "agent" is used to refer to objects that behave differently in different situations. Agents interact with other agents in a distributed environment in order to achieve goals. Meanwhile, each agent builds "models" reflecting other agents' properties. The model helps an agent to divine the outcome of future interactions. The identification and authentication in an agent & model-based authentication are defined as follows [89]:

- *Identification* is a process that selects one or more models from agent's model space.

TABLE XVII
SUMMARY OF AUTHENTICATION MECHANISMS

Mechanism		Based on	Advantages	Disadvantages
Cryptography-Based	Symmetric	• Shared secret key	• Knowledge of the shared secret key allows a principal to encrypt and decrypt arbitrary messages	• Vulnerable to replays • Not practical on a large scale
	Asymmetric	• Public key and private key	• Protects confidentiality • Assures receivers of the identity of the source message • Adapts to network size and service requirements	• Problem of public key distribution
Router-Based		• Routing restrictions associated with workstations	• Most basic form of authentication • Can be used in conjunction with other methods	• Coarse level of protection • Does not scale well • Vulnerable to a determined attacker
Agent & Model-Based		• An agent matching objects with a model from the object's model space	• Overcomes problem of public key distribution	• Object could match no models or more than one model

- *Authentication* of an object against a selected model is a process that decides whether the model matches up the object. The model that we authenticate the object against was chosen from the model space during identification.

An agent can identify a particular model through three different means: names, associations, and constraints. This authentication method is based on the assumption that an agent can match objects with models recorded in the model space. However, problematic scenarios can result if an object matches no model or more than one model.

A summary of the authentication methods discussed is shown in Table XVII.

3.3 Multidatabase Authentication Policies

In Section 3.2, we discussed the authentication mechanisms commonly used in distributed systems. On top of each mechanism, multidatabase designers can apply different policies to satisfy the security needs of a particular system. In this subsection, we present and compare several multidatabase (federated database) authentication policies. When discussing the security issues in centralized database systems, we mentioned that the correct authentication of users is an important task because it is the necessary prerequisite for access control [1]. The authentication problem is more complex in multidatabase systems as shown in Table XVIII [91]:

In federated systems, access to data can be seen at two different levels: the federation level and the local level [71]. At the federation level, users explicitly require access to the federated data, while at the local level, the local requests corresponding to the global requests must be processed. With respect to who should enforce authentication, we can distinguish between local and global authentication. Table XIX summarizes the pros and cons of each approach. In *local authentication* [90] users are required to re-authenticate themselves at each local site. Upon reception of a request by the federation, the local site asks the user to identify himself/herself and, after authentication, performs access control and possibly returns the data to the federation. In *global authentication* [91], a user's identity is passed to the site by the

TABLE XVIII
ISSUES INCREASING THE COMPLEXITY OF AUTHENTICATION IN MULTIDATABASE SYSTEMS

Heterogeneity of Local Authentication Components	• Global users have to pass through different procedures to gain access. • The identity of a user can vary from system to system. • Each user should be authenticated once but correct to all relevant participating systems per session.
Local Autonomy	• Component DBSs have the autonomy to decide whether a user is valid.
Uniformity	• The same user may have different identities and identifiers but has to be handled uniformly.

TABLE XIX
ADVANTAGES AND DISADVANTAGES OF LOCAL AND GLOBAL AUTHENTICATION

Local Authentication	Advantages	• Local access decisions can be taken with respect to identifiers known at the site. • Does not require the local site to be informed about remote or federation identities of users.
	Disadvantages	• May make access control process very heavy. • Each access request on federated data could be split into several access requests on local data (possibly stored at different sites), which would require the user to login to each site involved in the transaction.
Global Authentication	Advantages	• Users are not required to authenticate themselves at each local site.
	Disadvantages	• The local system needs to put some trust on the remote or federation identity. • Authorization at local sites needs to be specified with respect to identities not administered by the local site itself.

federation along with the request. Authentication at the local level can therefore be enforced by considering:

- The federation from which the request arrived,
- The identity of the user at the federation, or
- The remote identity of the user at the site from which he/she is connected to.

In the first case, authentication decisions are made only with respect to the federation. In the latter two cases, the local site makes identity-based access decisions with respect to the user's identity.

Jonscher and Dittrich provided three authentication schemata [75] that later on extended by Hildebrandt and Saake [91]. The authors assume that each local system associates each of its users with exactly one identity and one identifier. Users can be classified into three groups:

(1) Local users with one identity per affiliated system,
(2) Global users with one global identity, and
(3) Federated users with local and global identities.

Hildebrandt and Saake proposed three authentication policies: direct, indirect, and global [91].

Direct Authentication. Direct authentication requires the user to be authenticated by all participating systems that he/she wishes to access. This approach is suitable under the following situations:

- High local autonomy and security requirements.
- Low trust between the participating systems.
- Invincible heterogeneity.

In the absence of a global authentication component, users operate only with their local identities. An application is required on every possible login system to establish a connection to the authentication components of other systems. Users must directly log on to all systems where hardware based authentication components are in use.

If a global authentication component is in use, it can authenticate the global identity of users and pass the authentication information to CDBSs according to an association table that records the mapping between a global identity and all local identities. This allows a user to first log on to a local system and then receive access to the others through the global component. A user should not be allowed to operate simultaneously at different participating systems without global authentication. In this approach, global users can only receive local access through the acquisition of local identities.

Indirect Authentication. The indirect authentication approach delivers the relevant user information for the local authentication indirectly from a special component, not directly from the user. Without a global component, each database stores not only a user's identity and identifier used by that database, but also the user's identities and identifiers used by all other databases. For example, say a user logs on to database A and wants to access database B. Database A would then pass the user's identity and identifier to database B. The main difficulties of this approach include managing the trust between the systems to enable the mutual storage of identifiers and ensuring that the participating systems support the same security standard. In the presence of a commonly trusted global component, a user can be authenticated using his/her global identity and identifier. This approach allows the possibility of granting access to global users without local identities. The local systems provide some identities to the FDBMS that are attached with different but precisely defined access rights. The global component can now associate these identities with global users, giving these users limited and indirect access to local data. Nevertheless, local administrators have the right to decide which user receives access. There are two ways to ensure this:

- Local administrators could establish a list of persons who may not access the system. The FDBMS must abide by this list, alternatively.
- The FDBMS is required to obtain local approval for each decision.

The global component assisted indirect authentication method has the following requirements:

- The global management of local identifiers needs the trust of local systems.
- The propagation of local identifiers and their delivery for local authentication leads to high requirements for a secure data transfer.
- The association of identities needs the agreement of the user and of the relevant local administrators.

Global Authentication. The global authentication approach allows the FDBMS to take full control of the authentication process. This approach is only suitable for special applications since it sacrifices local autonomy.

3.4 Access Control in Multidatabase Systems

Research issues involved in access control in multidatabase system include administration of authorization, authorization specification, and access control policy heterogeneity [71]. In this subsection we will discuss the general issues and then compare several proposed multidatabase access control models.

3.4.1 Administration of Authorization

In a multidatabase system, there are objects that belong to only CDBMSs, objects created at the federation level, and objects imported from the CDBMSs to the federation. For objects created directly by the global component or that belong only to CDBMSs, classical administrative policies developed for centralized system can be applied. However, managing authorization for imported objects is more complex. In practice, three approaches are often considered [71]:

- Delegating the administration of the objects to the federation administrator.
- Leaving the privilege of specifying authorizations to the administrator of the local object.
- Allowing both the federation administrator and the local administrator to specify authorizations.

The different approaches imply different degrees of administrative burden on the user. The authorization mechanism should be able to enforce different policies according to application requirements. Table XX classifies and examines the different levels of authorizations possible in multidatabase systems.

TABLE XX
AUTHORIZATION LEVELS

Authorization Type	Description
Full authorization	• Local access decisions are based on local user identities. • Is cumbersome since the user has to enter a number of passwords to authorize a request made on multiple local systems.
Medium authorization	• Some trust (e.g., global authorization server) is placed on the global system that correctly validates the authority of users. • The global system authorizes a request and associates an identifier with every component request obtained by decomposing the global request. • The identifier is then used by local systems to make access control decisions. • The global and local systems cooperate to provide authorizations of requests.
Low authorization	• The global system alone authorizes requests for data from external users and the component requests are directly executed at local systems. • Is very convenient if the system is to contain only global users. • Requires considerable amount of trust in security mechanisms of the global system.

3.4.2 Authorization Specification

When forming a multidatabase system, one important decision to make is how to specify authorization rules at different levels (global level and local level) and how to resolve the conflicts. Three basic approaches are seen in the literature:

(1) *Independent approach*. This first approach considers the authorization specified at the global level and the local level as independent from each other. The federation administrator and the local administrator specify their rules independently on federated data objects and local objects, respectively. However, the two administrators must cooperate in order to avoid inconsistent specifications. This approach is unrealistic and against the whole purpose of federation.
(2) *Top-down derivation*. Access authorizations are specified at the global level and then derived at the local level [75]. The global administrator specifies the rules for a user to access global objects. Local data objects involved in the authorization are determined and the authorization requests are derived from the global authentication rules. If there is an inconsistency between the authorization specification defined at the local level and the derived authorization request, the authorization is rejected at the global level. This approach is very difficult because in many occasions it is impossible to precisely map a global subject to a set of local identifiers.

(3) *Bottom-up derivation*. Authorizations at the global level are derived form authorizations at the local level [92]. In this approach, when an object is imported into the federation, its global authorizations are derived from its local authorizations. Authorizations defined for an object may differ in different local DBMSs. When conflicts arise, no global authorization can be derived for that object. Mapping subjects at the global level to local level is still a problem in this approach. Moreover, when changes are made at the local level, maintaining the consistency between the two levels is problematic.

3.4.3 Access Control Policy Heterogeneity

Access control policy heterogeneity refers to different local sites enforcing different access control policies. A local DBMS may adopt one of the variations of the access control polices we discussed in Section 2: mandatory access control (MAC), discretionary access control (DAC), or role-based access control (RBAC) policies. Heterogeneity may arise even if all sites enforce the same type of policy. Table XXI shows how heterogeneity may occur when using different access control policies.

3.4.4 Various Multidatabase Access Control Models

In this section, we briefly review several access control models proposed over the years. This will help us understand how different aspects of access control are handled in different systems. Table XXII compares the advantages and disadvantages of these models.

Wang and Spooner [94] proposed an approach to enforce content-dependent access control in a heterogeneous federated system. In such systems, a user must register at every local site that he/she needs to access. Authorizations can be specified at both the local and global levels through view materialization. As far as the administration of authorization is concerned, they enforce ownership-based administration. Local autonomy is preserved by giving the local administrator the rights to decide whether a view materialization request from the global level should be granted.

TABLE XXI
ACCESS CONTROL POLICY HETEROGENEITY

Policy	Heterogeneity May Occur If Different Sites:
Mandatory	• Use a different granularity of classification • Refer to different classification lattices • Give different meanings to the same security levels
Discretionary	• Allow different types of authorizations to be specified (i.e., one site may enforce a closed policy while another site enforces an open policy) [93]

TABLE XXII
SUMMARY OF ACCESS CONTROL POLICIES

Policy	Advantages	Disadvantages
Wang and Spooner	• Allows local systems to preserve authorization autonomy	• Authorizations can be specified only for users • A user must be registered at any local system he/she needs to access
Mermaid	• Preserves authorization autonomy • Supports different degrees of authentication autonomy	• Does not support decentralized authorization at the global level • Users must be registered with Mermaid and local systems • Access control is based on Access Control Lists
Jonscher and Dittrich	• Supports different degrees of authentication autonomy • Decentralized administration • Preserves local autonomy	• Does not allow local systems to share their objects with reference to specific privileges • User can access an object only if an authorization is granted by both the global owner and the local administrator
Blaustein et al.	• Very flexible • Local autonomy not restricted	• Heavy burden on users responsible for negotiation at each site
Vimercati and Samarati	• Both federation and local sites are involved • Local systems do not need to keep track of identifiers for each single user of the federation	• Mapping a global authorization into a set of local authorizations can be difficult.

In Mermaid [90], a front-end system is used to integrate multiple homogeneous DBMSs. Authorizations are specified both at the global level and the local level, but the access control decision is always made locally.

Jonscher and Dittrich [75] proposed a model allowing authorization to be specified at both the global and local levels. In this model, a global security administrator specifies the local identities corresponding to each global identifier. A global authorization is generated only if all corresponding local authorizations can be granted.

Blaustein et al. [95] introduced the concept of agreement into control access in federated database systems. Agreements are rules regulating the access to the cooperating database systems by users connected from the different sites. Two kinds of agreements are considered: *action agreements* and *access agreements*. Action agreements describe the action to be taken in response to database requests, while access agreements allow enforcing exceptions to prohibitions otherwise in effect. The identity of users at the remote site from which they submit the request is used in access control.

Vimercati and Samarati [71] proposed an access control model where both the federation and local sites are involved. Each request on an imported or composite object is translated into a request or set of requests on the corresponding local objects. Each of these requests must be communicated to the appropriate site for access control because local authorization must be present for the data in the local objects to be released. Additionally, data is not replicated at the federation so it must be obtained upon each request. The mapping of operations on federated objects to operations on the corresponding local objects is enforced by the data management system of the federation. Then, the federation sends each site storing a local object involved in the transaction an access request for the groups to which the user belongs and the remote identity of the user. In the case of local transactions the user will need to re-authenticate himself/herself at the local site. Each local site will check the local authorizations and grant or deny the access according to the policy established for the object. In particular, in the case of site-retained or cooperative policy, access will be granted if an authorization exists for the access and no negative authorization exists. In the case of federation-controlled administration, access will be granted if no negative authorization exists. The final reply of the federation to the user is the result of the replies to the local requests. In summary, global access is granted if all local sites accept the local requests; it is denied otherwise.

3.4.5 An Example—Authorization Model for the Summary Schemas Model (SSM)

A global authorization model for SSM based on Role Based Access Control (RBAC) paradigm was proposed in [96]. The main idea is to map an individual subject in local databases to a common role defined at MDBS level and to tag access terms in the SSM hierarchy with a set of roles allowed to access those terms. Subjects and objects are specified both at local databases and at the MDBS level. At local databases, there are local subjects and local objects. Local subjects, as the name suggests, are defined locally and manage local databases. Local objects are data sources created and maintained locally. Each local subject can access local objects according to local access control rules independent of other local databases. In addition, no assumption is made about authorization models used at local databases.

At MDBS level, access terms in the SSM hierarchy are global objects exported from underlying local schemas. Higher terms are populated in a hierarchical structure according to their word relationships. Thus, there is no composite object at MDBS level. Since global subjects are allowed to access objects across multiple local databases, it is natural to assume that only a subset of local subjects is allowed to be global subjects. However, mapping individual local subjects to global subjects is a tedious and error prone task. Using roles as global subjects certainly helps simplify

the task. In addition, local databases are responsible for mapping their local subjects to corresponding global roles. A local database may maintain a table that keeps track of which subject is mapped to which global role. If a new role is added or an existing role is deleted, all local databases will be informed and their local subjects can be remapped. Nevertheless, frequent changes of global roles are not anticipated. Moreover, when a user logs in at any node, the authentication can be done at a local database where a user has an account. Hence, no global authentication is needed.

Imposing authorization information to the SSM adds both space and time overhead and clearly affects the query resolution of the SSM. The performance of the proposed model is evaluated and compared with that of the original SSM. The simulation results showed that the proposed model outperforms the original SSM model since user queries with insufficient authority are rejected earlier. The early rejection of unauthorized queries reduces the network traffic and workload at both SSM nodes and at local databases. Thus, the response time of valid queries in the proposed model is lower than that of the original SSM.

The SSM platform was extended to include mobility and wireless communication [76]. A mobile computing environment introduces extra requirements such as disconnection and limited bandwidth to the model. In Section 4.4, we consider security issues in the mobile environment within the scope of the SSM.

3.5 Inferential Security

As noted in the literature, the current developed database frameworks share a single trait that is termed as *the principle of paranoia* [97]—The DBMS must take all steps necessary to insure that the user u cannot infer any item in a pre-designated set $S(u)$ of items that are to be kept secret. Thus, it is possible for external users to infer information in the information repository even when enough access rights are available. This is a security breach and needs to be addressed as early as possible when a query is submitted. For example, James may/may not have a criminal record. This record has restricted access if it exists and information cannot be publicly available. If an unauthorized person wants to determine if James has a criminal record and would be happy with a "Yes/No" answer, he/she could query the database for non-availability of James' information. The query result would return "True/False" instead of failing since information about James is not being accessed. Thus, the unauthorized person can infer that James has/has not a criminal record even though the details were not retrieved and James' record was not accessed.

3.6 Integrity Issues

Additional inter-database integrity constraints could be required after integration of component databases in a federated system. However, a crucial aspect of federated

database systems known as site autonomy may allow local operations to violate these global inter-database integrity constraints. These local operations violate global integrity because they are performed outside the control of the federation layer. Thus, enforcement of global integrity constraints does not necessarily always guarantee that the database is consistent or that no integrity violations occur. A federated database is consistent if all global[2] integrity constraints as well as all integrated (local)[3] integrity constraints hold.

Global integrity constraints are sub-divided into global key constraints, global referential integrity constraints, and global aggregate constraints. Enforcing these integrity constraints results in a certain reduction of the local autonomy [98] (Table XXIII).

The authors of [98] proposed to integrate active rule paradigm in order to realize the global integrity enforcement in federated[4] database systems. Active rules need to be used to initiate appropriate integrity preservation or restoration operations when detected or signaled for possible security violations. Possible security violations could result from violations in transaction[5]/database[6] consistency.

Global integrity constraints and possible integrity restoration actions can be informally specified as rules as follows:

Define rule *Global_Integrity_Rule*
On event which potentially violates the integrity of the federated database
If a global condition formulated over the state of the federated database is true
Do global action(s) that restore(s) a consistent federated database state

TABLE XXIII
GLOBAL INTEGRITY CONSTRAINTS

Global Constraint	Description
Global key constraints	• Ensures the uniqueness of an object of the federated schema
Global referential integrity constraints	• Used to describe relationships between two objects from different local databases
Global aggregate constraints	• Used to model constraints on multiple objects in different local databases

[2]At federation level, there exist global integrity constraints defined on federated database schemata.

[3]At the component system level, there exist local integrity constraints defined on the local database schemata.

[4]A federated database system integrates heterogeneous component database systems.

[5]*Transaction consistency* denotes the correctness of interleaved transactions executions in multi-user environment.

[6]*Database consistency* refers to correctness of data according to the specified semantic-related integrity constraints. These integrity constraints restrict the possible database states and database state transactions, respectively, to valid ones.

The local components can trigger corresponding global integrity constraint rules as defined above whenever a particular (re)-action is sought to restore consistency of the database. The format of a local rule is as follows:

Define rule *LocalRule*
On local event
Do trigger corresponding global rule(s)

4. Mobile Data Management

In Sections 2 and 3, we discussed security issues in traditional databases and multidatabases, respectively. In this section, we turn our attention to mobile systems. In mobile systems, the goal is for clients to access data "anytime, anywhere." Introducing mobility to a system brings new challenges to the design, but the underlying security requirements still remain the same. Traditional security solutions from the "sometime, somewhere" environment must either be extended, taking into account the limitations imposed by mobility and wireless communication, or new solutions more suitable for the "anytime, anywhere" environment must be devised. We begin this section with a description of mobile architectures and discuss data access techniques. Then, we present some of the new constraints introduced by mobility. We also present an extension to the work described in Section 3.4.5, which adds mobility to the SSM. Finally, we discuss pervasive computing—its applications and the security and privacy requirements of these emerging systems.

The design of mobile information systems is complicated by several constraints, which are intrinsic to mobility [99]. First, mobile elements are resource-poor relative to static elements. Mobile devices must be small, lightweight and consume limited amounts of power. These limitations reduce the computational resources (processor speed, memory size, disk capacity) of mobile elements compared to static elements. Additionally, the flexibility and ease of use of display modules and input devices are limited by these constraints. Second, mobile computing devices are more vulnerable to loss or damage. Third, mobile connectivity varies in performance and reliability. Some buildings offer reliable, high-bandwidth wireless connections while others may only offer low-bandwidth connections. Outdoors, a mobile client may have to rely on a low-bandwidth network with gaps in coverage. Finally, mobile devices rely on finite energy sources. Concern for power consumption spans many levels of hardware and software. Wireless communication is characterized by several limitations, including high error rates, variable delay, and inconsistent performance. These issues must also be considered when designing mobile information systems. Current mobile network architectures can be categorized as shown in Table XXIV [101].

TABLE XXIV
MOBILE NETWORK ARCHITECTURES

Architecture	Description
Cellular Networks	• Provides voice and data services to users with hand-held phones. • Continuous coverage is restricted to metropolitan regions. • Movement over a wide area may require the user to inform the network of the new location. • Low bandwidth for data-intensive applications. • Could be based on either analog technology or digital technology.
Wireless LANs	• A traditional LAN extended with a wireless interface. • Serves small, low-powered portable terminals capable of wireless access. • Connected to a more extensive backbone network, such as a LAN or WAN.
Wireless Area Wireless Networks	• Special mobile radio networks provided by private service providers (RAM, ARDIS). • Provides nationwide wireless coverage for low bandwidth data services, including e-mail or access to applications running on a fixed host.
Paging Networks	• Receive-only network. • No coverage problems. • Low bandwidth. • Unreliable.

Satellite Networks	• Unlike the static ground Mobile Support Stations, satellites are not fixed.
	• Normally Classified based on their altitudes (from earth) into three classes:
	  • Low Earth Orbit Satellites (LEOS),
	  • Medium Earth Orbit Satellites (MEOS),
	  • Geostationary Satellites (GEOS).

In a cellular network (Fig. 24), a mobile subscriber's (MS) service profile, a database entry that includes information about the services the user receives, is stored at his/her Home Location Register (HLR). The HLR is permanently assigned based on the user's mobile telephone number. As the user moves throughout the network, his/her service profile is downloaded to a Visitor Location Register (VLR), a database that stores temporary information about MSs. Typically, a VLR is combined with a Mobile Switching Center (MSC). A MSC performs standard telephony switching functions as well as functions necessary to handle a MS (registration, authentication, location updates, etc). The gateway MSC connects the cellular network to some other type of network (i.e., ISDN or wireline) [102]. When someone places a telephone call to a MS, the gateway MSC asks the HLR to find the MS. In turn, the HLR

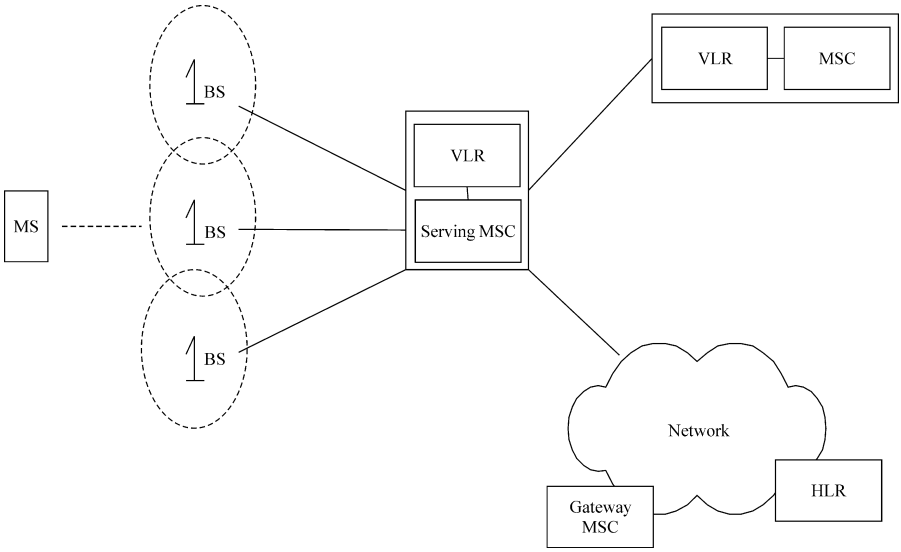


FIG. 24. Cellular network.

asks the MSC/VLR to find the MS. Then, the MSC/VLR contacts the appropriate Base Station (BS), which pages the MS on a radio interface.

Data can be accessed in a mobile client/server architecture (Fig. 25) or an ad-hoc network (Fig. 26) [74]. In this chapter, we are primarily concerned with the mobile client/server architecture. In this paradigm, a small number of trusted server sites constitute the home of data [99]. A large number of untrusted client sites can efficiently and safely access this data. Techniques such as caching and read-ahead can be used to improve performance, while end-to-end authentication and encrypted transmission can be used to preserve security.

In ad-hoc networks, a collection of wireless mobile nodes form a temporary network without the aid of centralized administration or standard support services. Information is available at many participating nodes and is no longer owned by trusted servers.

4.1 Mobile Data Access in the Client/Server Architecture

When accessing remote data, mobile clients face widely varying and rapidly changing network conditions and local resource availability [100]. To allow applications and systems to continue operating in this dynamic environment, the mobile

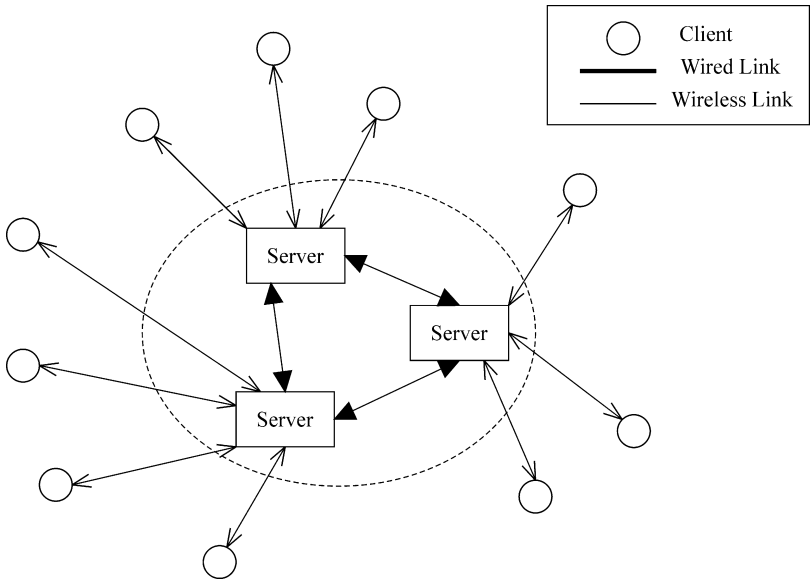


FIG. 25. Mobile client/server architecture.

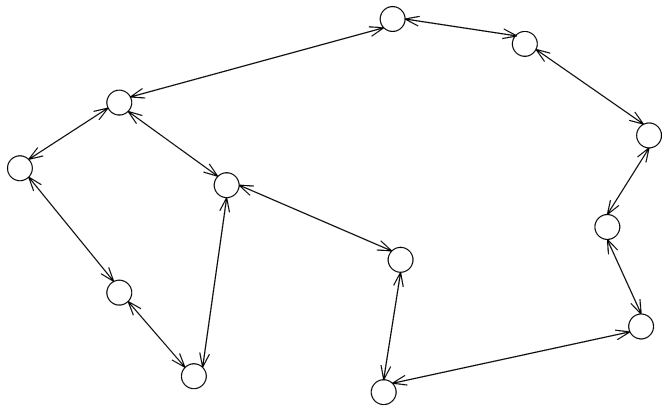


FIG. 26. Ad-hoc network.

client/server system must react dynamically and adapt the level of computation between the mobile and stationary nodes. The resource limitations of clients may require that certain operations normally performed on clients be performed on a server.

TABLE XXV
MODES OF DATA ACCESS IN CLIENT/SERVER MODEL [100]

Classification	Description
Server push	The server push delivery is initiated by server functions that push data from the server to the client.
Client pull	The client pull delivery is initiated by client functions which send requests to the server and "pull" data from the server in order to provide data to locally running applications.
Hybrid	The hybrid mode makes use of server push and client pull based on the situation arising.

Conversely, because connectivity may be uncertain, clients must sometimes emulate server functions.

The range of strategies for adaptation is delimited by two extremes—laissez-faire and application-transparent [99,100]. The laissez-faire adaptation approach lacks a central arbitrator to resolve incompatible resource demands of different applications and to enforce limits of resource usage. This approach also makes applications more difficult to write and fails to amortize the development cost of support for adaptation. At the other end of the spectrum, the application transparent adaptation approach is attractive because it is backward compatible with existing applications. The system is the central point for resource arbitration and control. There may, however, be situations where the adaptation may be inadequate or even counterproductive. The application-aware adaptation approach allows applications to determine how best to adapt to varying conditions, but also allows the system to monitor resources and enforce allocation decisions as necessary.

Data access strategies in the mobile client/server model are characterized by delivery modes, data organizations, and client cache consistency requirements. Table XXV classifies the different delivery modes and briefly describes them.

Data organizations include mobility-specific organizations, including mobile database, fragments in server storage and data multiplexing, and indexing in the server-push delivery mode [100].

4.2 Limitations of the Wireless and Mobile Environment

In addition to the basic constraints of mobility, there are several other problems inherent in the mobile environment, including frequent disconnections, limited communication bandwidth, and heterogeneous and fragmented wireless network infrastructures (Table XXVI).

TABLE XXVI
LIMITATIONS OF A MOBILE ENVIRONMENT [74]

Limitations	Concerns/Side Effects
Frequent Disconnections	• Handoff blank out[a] in cellular networks. • Long down time of the mobile unit due to limited battery power. • Voluntary disconnection by the user. • Disconnection due to hostile events (theft, destruction). • Roaming-off outside the geographical coverage area of the window service.
Limited Communication Bandwidth	• Quality of service (QoS) and performance guarantees. • Throughput and response time and their variances. • Efficient battery use during long communication delays.
Heterogeneous and Fragmented Wireless Network Infrastructure	• Rapid and large fluctuations in network QoS. • Mobility transparent applications perform poorly without mobility middleware or proxy. • Poor end-to-end performance of different transport protocols across network of different parameters and transmission characteristics.

[a] A handoff blank out happens when a mobile client is crossing a cell and the new cell is not able to accommodate the client call.

4.2.1 Frequent Disconnections

Different degrees of disconnections, varying from total disconnection to weak disconnection, may exist. Weak disconnections, or narrow connections, occur when a terminal is connected to the rest of the network via a low bandwidth wireless channel [101]. Caching required files before disconnections is a possible solution; however, disconnections would need to be predicable. Handoffs can also lead to disconnections in cellular networks if the destination cell cannot accommodate a new request. Three strategies exist to detect when a handoff is required as shown in Table XXVII.

In addition, the handoffs can be of two types as examined in Table XXVIII.

4.2.2 Limited Communication Bandwidth

The concept of bandwidth and energy management plays an important role in the design of wireless information services where mobile clients will query and possibly update remote databases located on fixed network nodes through a wireless channel. The cost measures for accessing and updating data are dependent on bandwidth and energy availability.

Generally, a user queries for information stored in server databases over bandwidth limited wireless channels. This information can be supplied to the user in two different ways as listed in Table XXIX.

TABLE XXVII
HANDOFF DETECTION TECHNIQUES [128]

Strategy	Description
Mobile phone-controlled handoff	The mobile phone continuously monitors the signal from the surrounding base stations, and initiates the handoff when some handoff criteria are met.
Network-controlled handoff	The surrounding base-stations measure the signal from the mobile phone, and the network initiates the handoff when the criteria are met.
Mobile phone-assisted handoff	The network asks the mobile phone to measure the signal from the surrounding base stations. The network makes the handoff based on the report from the mobile phones.

TABLE XXVIII
SCENARIOS FOR HANDOFFS [128]

Type	Description
Inter-Base Station	• A user moves between base stations connected to the same Mobile Switching Center (MSC).
Inter-System Handoff	• A user moves between base stations connected to different MSCs. • This handoff incorporates additional responsibilities and complexities.

TABLE XXIX
INFORMATION RETRIEVAL METHODS [128]

Type	Description
Data Broadcasting	• Data is broadcast on the channel periodically • Accessing broadcasted data does not require uplink transmission and is "listen only"
Interactive/On-Demand	• The client requests a piece of data on the uplink channel • The server responds by sending this data to the client on a downlink channel

Query capacity is defined as the number of queries that can be handled by the server per unit time. To maximize available bandwidth utilization, the query capacity should be increased. However, increasing the number of queries at an individual mobile unit would result in more energy consumption. Thus, a trade off needs to be made between increasing query capacity and decreasing energy consumption. Data broadcasting information frequently accessed by users is a possible solution that increases query capacity and decreases energy consumption. The server initially

broadcasts the directory of information to the clients and then starts the actual broadcasting of the information. The client thus saves energy by avoiding transmissions and waking up from suspend modes during a broadcast only when necessary.

To reduce broadcast length, access latency, and power consumption, broadcasting along parallel air channels is employed [103]. A scheduling algorithm to minimize response time and power consumption when retrieving objects from indexed parallel broadcast channels was proposed in [104]. One of the main problems in a parallel channel environment is the possibility of conflicts while pulling objects from the air channels. A conflict occurs when more than one requested object is transmitted at the same point of time on different parallel channels. Since mobile units can only tune into one channel at a time, the retrieval process has to wait for the next broadcast cycle(s) to download the remaining requested objects [105,106]. Conflicts affect the access latency and hence the response time and power consumption. In order to predict and analyze the impact of the conflicts, statistical foundations could be used to calculate the expected number of broadcast cycles required to retrieve a set of objects. Two heuristics were used to analyze the effect of conflicts on object retrieval [105,106]. The Next Object heuristic always retrieves the next available object in the broadcast, while the Row Scan strategy reads all the objects from one channel in each pass. Previous studies showed that when the number of requested objects exceeds 45% of the total objects, the Row Scan heuristic provides a better solution; moreover, it reduces the delay associated with switching between channels.

The retrieval protocol proposed by [104] attempts to produce an ordered access list of requested objects that reduces:

- The number of passes over the air channels, and
- The number of channel switching.

The retrieval scheme based on a set of heuristics determines the sequence of objects to be retrieved in each broadcast cycle. It attempts to minimize the energy consumption of the mobile unit and the total number of passes required to satisfy the user query. The schedule is determined based on the following three prioritized conditions:

(1) Eliminate the number of conflicts.
(2) Retrieve the maximum number of objects.
(3) Minimize the number of channel switching.

Simulations showed that the new scheme reduces both the response time and the number of passes compared with the Row Scan heuristic. Additionally, this method reduces energy consumption when a reasonable[7] number of objects were requested.

[7]Less than 5% of the total broadcast data.

TABLE XXX
GROUPING SECURITY ISSUES

Basis	Security Issues
Security goals and associated threats	• Confidentiality • Integrity • Availability (including accountability and non-repudiation)
	• Information leakage • Integrity violation • Denial of device • Illegitimate use • Unaccountability
Communication problems	• Content privacy • Location privacy • Unlinkability of sender and recipient
Objects to protect [108]	• Information • Meta-data
Characteristics [107]	• Hardware • System • Application

4.3 Security Issues in Mobile Environments

Security issues in mobile systems have been grouped in the literature [107,108] based on a variety of factors as shown in Table XXX.

4.3.1 Information and Meta-Data Protection

Information is comprised of requests made by mobile clients and the requested data that is transported back to mobile clients from servers. Meta-data consists of user profiles, information about the current resource situation, information characteristics, location, and time. Security needs to be provided to the information and meta-data during

- Transfer, and
- Management and access.

Some solutions to the problems described in Table XXXI include:

- The database system should be responsible for avoiding loss of data in case of unexpected disconnections with the help of transaction recovery,
- Cryptography could be used to prevent masquerades or provide identity masking,
- Use of asymmetric encryption for user authentication and symmetric encryption for secure communication,

TABLE XXXI
SECURITY ISSUES OF INFORMATION AND META-DATA [108]

Type		Description
Information Security Issues	Transfer	• Disconnections occur often in wireless communication. This situation can endanger data consistency. • The higher frequency of network partitioning in wireless systems requires more powerful error recovery than fixed networks. • The possibility of attackers to masquerade the mobile unit or base-station. • Wireless links facilitate eavesdropping, which is very hard to detect.
	Management and Access	• Loss of mobile units is a possibility leading to loss of data and confidentiality. • Scarce resources may cause faulty situations. The system may not be able to or the user may renounce from carrying out security methods. • Disproportion between the amount of requested data and the available resources leading to violation of availability or integrity.
Meta-data Security Issues	Transfer	• Current whereabouts and movement of user are matters of privacy and ideally should be known only to the user herself/himself. • All user identification information including message origin and destination needs to be protected. • When a user crosses cell boundaries, his/her information will be transferred and replicated to the adjacent Base Stations. Risks for very sensitive personal data are increased due to multiplication of the points of attack and possibly different trust levels between nodes.
	Management and Access	• Heterogeneity of access control models (multilevel, discrete, role-based) across base-stations need to be taken into consideration. • Heterogeneous integration of data in homogeneous models (apart from heterogeneous security aims and strategies) also needs to be considered. • User whereabouts and movements can also be deduced in indirect ways. A mobile user working on databases accesses data necessary in his/her computing environment. The data which the user has accessed (created, modified and read) at which time make a deduction of his movements possible because of the location dependencies of data.

- Register mobile users with their real identity or pseudonyms with that domain's authentication server,
- Use of pseudonyms or aliases for maintaining anonymity,

- Identity of users need to be kept secret from service providers,
- Prevention against traceability of network connections in mobile can be offered through either MIXes or the Non-Disclosure Method [108], which use cryptography.

4.3.2 Hardware Characteristics

The limiting hardware components in a mobile computing environment are the mobile access devices and the wireless medium. Various limitations and characteristics of these components need to be taken into consideration when providing a secure wireless transaction. These limitations are [107]:

- Mobile units range from simple PDAs to powerful notebooks. These mobile units are typically low power units with scarce computing resources. Low power and scarce computing resources was initially the motivation for making security optional in wireless environment.
- Wireless media is inherently less secure. It is comparatively easier to snoop or jam radio signals. This inherent property requires security to be moved to the lowest communication layer to prevent snooping or eavesdropping. However, preventing radio signals from being jammed is difficult.
- Wireless bandwidth is typically orders of magnitude less than wired bandwidth. Hence, providing security requires reducing the number of messages in the network.

In the physical world it is easy to prove payment using cash and credit cards. On the Internet (via a wired or wireless medium), it becomes harder to confirm the identity of the person attempting to complete a purchase. One solution may be biometric identification (Section 2.2.4). For example, a fingerprint scanner could be built into wireless devices. The scanned image would be transmitted to the merchant who would check it against a secure database of fingerprints. If the user has registered in this database and the information matches, the merchant would allow the purchase to proceed.

Some other authentication techniques implemented in hardware in wireless handsets are described in [109]. The techniques include:

- Security is integrated into terminal hardware.
- Security functionality is installed on SIM cards.
- Additional security chip (dual chip) is provided with SIM card.
- Integrated reader for external SmartCards “dual slots.”
- External reader for external SmartCards.

4.3.3 System Characteristics [107]

A number of infrastructure characteristics are important for providing security. These characteristics include autonomy, mobility, and time stamping. The following characteristics need to be taken into consideration while providing security in wireless mobile environments:

- Autonomy is a critical issue in secure mobile communications. In LANs, assumptions are made that communicating end-points and the intermediate nodes are all parts of the same organization. Security is thus a relatively simple issue, given that there is a certifying authority authenticating the nodes. In WANs, this assumption fails, since frequently the communicating entities belong to different organizations, which are autonomously governed. It is possible to extend the schemes in LANs by providing a set of mutually trusting authorities.
- In the mobile environment, there is a new dimension added by mobility. Mobile units move between cells, and need to be authenticated upon entering each new cell. Currently, authentication requires communication with the HLR and this would lead to "across-the-globe" messages being exchanged. This would increase the network traffic in an already bandwidth scarce medium. A solution would be to make handoffs as efficient as possible eliminating the need to communicate with the HLR every time authentication is needed.
- Synchronization is currently employed in available security protocols in wired media. However, the assumption is invalid in a mobile environment as a mobile unit may travel across multiple time zones without changing its clock. An implicit form of time stamping needs to be developed in order to allow users to move between different time zones.

4.3.4 Application Characteristics [107]

As discussed earlier, applications in mobile computing environments may need to adapt to unstable network conditions. Secure multicast is an indirect consequence of mobility.

- Nodes in mobile environment can be dynamically addressed in order to protect location privacy.
- The majority of applications are for indoor wireless LANs. These would typically include classrooms sessions and meetings where there would be one speaker and many listeners. Support for a secure multicast would be thus an important aspect when designing security since we may not allow listeners to talk or speakers to listen.

4.4 An Example—Authorization Model for SSM in a Mobile Environment

The authors of [110] proposed an extension of the work described in Section 3.4.5 that meets the user demand for information to be accessible "anywhere, anytime." Broadening the scope of authorized SSM to accommodate mobility and wireless communication is not a simple task and adds to both time and space overhead. The main idea is to allow the clients to be mobile in nature and allow the local databases to be either mobile or stationary in nature.

To provide a secure wireless environment, a secure session must be initially set up between two parties (a client and base station/access point). The messages transmitted, after a session is set up, needs to be encrypted to prevent a third party from eavesdropping. Also, steps need to be taken to prevent non-repudiation. Based on this discussion, security involves the following four topics [111]:

- *Confidentiality*—transforming data such that it can be decoded only by authorized parties,
- *Authentication*—proving or disproving someone's or something's claimed identity,
- *Integrity checking*—ensuring that data cannot be modified without such modification being detectable, and
- *Non-repudiation*—proving that a source of some data did in fact send data that he/she might later deny sending it.

The AES (Rijndael) algorithm allows encryption of data thus verifying confidentiality and allowing integrity check facility. The Diffie–Hellman algorithm allows authentication of clients. Finally, SHA-256 provides for digital signatures in message exchanges during a session to prevent non-repudiation. These algorithms were used to set up a secure transmission link, and develop a handshake protocol as follows:

- Authentication or secure session setup is provided by password-based systems where a user is assigned a username and password to authenticate in the system. Diffe–Hellman Key Exchange Protocol [112] is used for authentication.
- Confidentiality and integrity checking is provided by encrypting messages during message exchange. Advanced Encryption Standard (AES) [113] is used for encryption of messages in the wireless network.
- Non-repudiation in secret-key algorithms is possible using Digital Signatures. In order to decide whether a user has sent a message, he/she includes a digital signature (formed by encrypting identity with the shared key) in the message. The digital signature is then decrypted at the receiver using the shared

secret key. Authorizations through digital signatures (message authentication code MAC) were provided using SHA-256 hashing utilities [114].

Thus, a completely secure session can be set up with these cryptographic algorithms. Before data transactions can be initiated, the two communicating parties must exchange messages to set up a secure channel.

- The client initiates the handshake by sending a **client_hello** message to the server. This message contains session ID, key refresh rate, private key encryption algorithm and its mode of operation, message authentication code (MAC) algorithm and a random number used to generate the encryption, and MAC keys from the master_secret.
- The server responds with a **server_hello** message accepting the security association proposed by the client, another random number (to be used during encryption key and MAC key generation), the **server_certificate** for authenticating itself to the client, its share $X = (g^a \bmod n)$ of the master_secret, where X is a large random number (**server_key_exchange)** and a **certificate_request** from the client.
- The client replies with its **client_certificate,** its share $Y = (g^b \bmod n)$ of the master_secret (**client_key_exchange)**, where Y is a large random number and a **certificate_verify** message to verify its certificate.
- Finally, the client and server exchange **change_cipher_spec** message to activate the session with the negotiated security association (the encryption algorithm and its mode of operation, the MAC algorithm, the session id and the key refresh rate) and a **finished** message to indicate successful key exchange.

After developing a test bed for the proposed solution, several experiments were performed to validate the proposed solution. As expected, acceptable performance degradation in query response time was experienced (Fig. 27). This was due to the nature of wireless communication and its characteristics—limited bandwidth, frequent disconnections, and unreliable medium of communication.

It was also observed that posting a query at a particular SSM level was a major factor contributing to the energy consumption during an active session. Energy consumed is mostly governed by number of session refreshes and idle mode time, which increases as SSM level increases. As a result, the energy consumed increases when queries are posted at higher SSM levels (Fig. 28).

This work could be extended by including modifications to minimize energy consumption. Also, the SSM could be developed for an ad-hoc network.

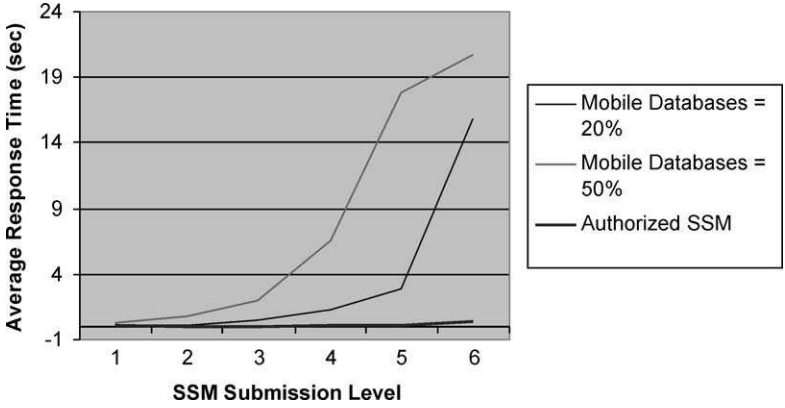


FIG. 27. Query response time.

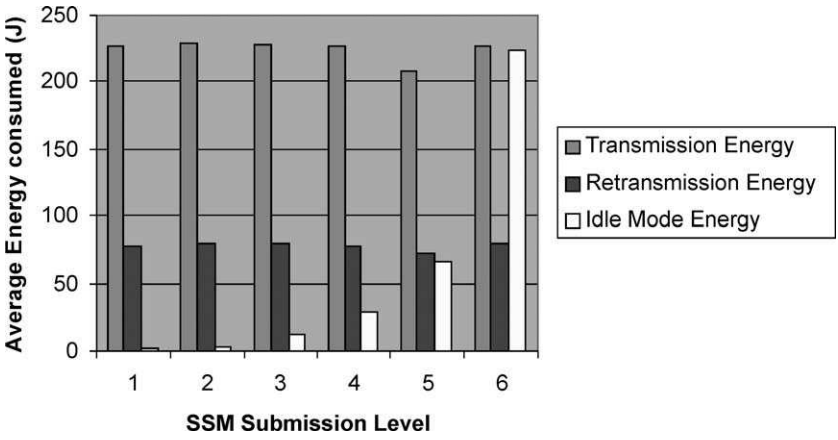


FIG. 28. Transmission, retransmission and idle mode energy consumed.

4.5 Pervasive Computing Systems

In "The Computer for the 21st Century," [115] Mark Weiser introduced the concept of pervasive computing as "a new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselves to vanish into the background."

In previous sections, we discussed security issues and solutions in both traditional databases and multidatabases. We also noted that security in a mobile environment

is a difficult problem and that many limitations must be overcome in order to design a secure system. So the reader can gain an appreciation for the security requirements in pervasive systems, we will begin the section with a brief description of potential applications. To facilitate the discussion, we begin by defining "smart dust" motes as self-powered devices with the ability to sense and communicate, which are contained in a single package [116]. A "smart space" is an area with the computing infrastructure embedded in the space [117]. For example, a number of intelligent sensors, or smart dust motes, could be distributed in a building and used to adjust temperature and lighting conditions in a room based on its occupants.

A sensor network is an ad-hoc network composed of a (potentially) large number of sensors that typically is connected to another network. Sensor networks have a wide variety of potential uses. Military applications include gathering battlefield intelligence, tracking troop movements, and detecting the use of biological or chemical weapons. A sensor's functionality varies with each application. For example, if a network was deployed to detect the use of biological or chemical weapons, the individual sensors might gather data and process the information before transmitting the result to a secure workstation. The workstation would then interpret the information collected within the sensor network and present the results to military officials. In other applications, sensors may gather and transmit data, but not do any intermediate processing. Sensor networks could also be used to detect pollution levels along beaches, to monitor conditions at volcanoes or fault lines, or to locate victims at disaster sites [118,119].

A second application of pervasive computing is the "smart office." In this case, the idea of a "smart space" is extended to an office building or campus and the range of computing devices includes PDAs, and desktop computers. For example, a user is preparing to give a presentation in another building on campus, but he/she will be late if he/she does not leave now. The system copies the file from his/her office computer to a PDA and then to the projector as the user moves from his/her office to the presentation room, allowing the user to make final changes to the presentation while in transit. The user starts presentation on time but as he/she continues, the system senses that some of the people in the room are unknown and warns the user, who decides to skip several slides that contain confidential information [120].

4.5.1 Security Issues

In both of these applications, we see that security is a necessary component of pervasive computing systems. Security must be considered at all levels of design, including hardware components, software applications, and communication protocols. The overall system must be secure. It is useless to protect wireless transmissions if an attacker could walk in the front door of the building and copy the file to a floppy disk

without detection. Likewise, developing and installing sophisticated face recognition systems is useless if wireless transmissions can be intercepted.

With respect to the wireless medium, the security goals include confidentiality, access control, data freshness, and data integrity [121,122]. Confidentiality seeks to prevent an eavesdropper from deciphering the message being sent. Access control aims to only allow authorized users to send and receive messages. Data freshness ensures that an adversary does not replay old messages and data integrity endeavors to prevent tampering with transmitted messages. The network itself must be robust so that the failure of a large number of devices cannot cause the entire network to fail [119].

Denial of Service (DoS) attacks are a concern in sensor networks because the information being collected may only be useful for a short period of time [119]. Each layer of the network is vulnerable to different types of DoS attacks. At the physical layer, wireless networks are prone to attacks that interfere with the node's radio frequencies, called jamming attacks. Using spread-spectrum communication is a traditional solution to minimizing the impact of jamming attacks, but this technique is not well suited to sensor networks because it requires additional power resources, which are already limited. If the attack is intermittent, nodes may be able to report the attack to the base station by sending high-power, high-priority messages. In the case of a sustained attack, the sensor network may be useless if the data is only meaningful for a short period of time. If the sensor network is operating over a large geographic area, a jamming attack across the entire network is unlikely. In this case, the nodes bordering the affected area could recognize the jamming attack and alert the base station. The base station could then use this information to compensate for the reduced functionality of the network. In a sensor network, almost every node will function as a router, which opens the system to attacks at the network and routing layers, including "neglectful node," "greedy node," and "black hole" attacks. A neglectful node will not forward messages on a random basis. Greedy nodes are neglectful nodes that also give priority to its own messages. In networks that use distance-vector-based routing protocols, black hole attacks are possible. In this scenario, a subverted node will advertise zero-cost routes to all other nodes, causing traffic to be directed to this node. To protect against attacks at the network/routing layers, several techniques can be employed. First, authentication techniques could be used to ensure that only authorized nodes are exchanging routing information. Unfortunately, this approach is not very practical in sensor networks due to the computational and communication overhead it incurs. A second technique uses redundancy to reduce the likelihood that a message will encounter an undermined node. Redundancy can be accomplished by using multiple paths or by using diversity coding. Diversity coding sends encoded messages along different paths and has a lower cost than sending multiple copies of the same message. Finally, nodes could monitor their neighbors for suspicious be-

havior. This approach matches the nature of sensor networks because the nodes must cooperate with each other and compensate for failures.

Maintaining the integrity of the mobile devices themselves is also important because the small size of mobile devices makes them susceptible to misplacement and theft. Commercial software applications are available from a variety of vendors to prevent unauthorized users from viewing certain data on a PDA [123]. PDADefense, for example, is an application that provides data encryption, bit-wiping bombs, decryption on demand, hardware button password entry, password masking, auto lock, and data transfer disabling. Bit-wiping bombs prevent brute-force attacks by bit-wiping all RAM locations after a specified number of attempts to unlock the device. Auto lock forces the user to reenter a password after a specified delay or when turning the device on. Infrared and hot syncing transfer operations are prohibited with data transfer disabling. For organizations, a system administrator can apply settings that meet the security and privacy policies of the organization. In particular, the system administrator can specify requirements for user passwords and selectively choose databases or applications to encrypt, preventing employees from modifying the security settings in a way that conflicts with the security policy of the company. In sensor networks, we want to prevent an attacker from altering a sensor node or obtaining sensitive information. If the node stops communicating with the rest of the network, it is impossible for the network to determine the cause. A module could fail entirely, lose power, or be compromised by an enemy. In any case, the module should erase its memory. The physical package of the module should be tamper resistant. In many cases, the tiny size of a sensor will limit the ability of an enemy to physically access the device. Camouflaging or hiding modes also provides a certain level of physical security [119].

4.5.2 Privacy Issues

Privacy is a challenging issue in mobile environments [101]. Users are concerned with privacy because information is accessible at any time. To protect privacy, it is necessary to develop sophisticated software to specify and enforce a user's personal profile. In this profile, users should be able to specify who is authorized to reach them and when and where they may be reached. A user may want to restrict the list of users who may "wake up" a mobile unit because receiving e-mails consumes battery power. Users who are concerned about privacy should consider the suggestions made by the Privacy Rights Clearinghouse [124].

In pervasive systems, privacy becomes an even more difficult. Location tracking is an important part of pervasive systems, but how this information is used must be carefully considered. Because the amount of data stored in a pervasive system far exceeds that of current computer systems, users must be able to trust that pervasive

system will not use the information it gathers in an unacceptable fashion. Similarly, the system itself must be sure of the user's identity and his/her privileges before allowing the user to access the system [117]. Within the context of pervasive systems, privacy is concerned with both the content and context (time and location) of information.

The concept of subscriptionless service architecture (SSA) has been proposed in [125]. It accounts for the instant and short-lived nature of mobile computing and the mobile user's interest in anonymity and privacy. The requirements of the SSA include client mobility, subscriptionless service access, anonymity, privacy, and several other traditional security concerns. Client mobility refers to the ability of a client to communication everywhere that an appropriate communication node is available. A client should be able to obtain information without a previous subscription; furthermore, a subscription will only be required if higher trust constraints (such as a credit card transaction) are needed. Anonymity provides that the mobile node will not expose any information that allows the user's identity to be determined. The main idea is to use pseudonyms to describe a user for a specified period of time. For a single transaction, a transaction pseudonym is used. Over longer periods of time, person and relationship pseudonyms are used so that the user can be recognized. Techniques for achieving privacy in pervasive systems are also discussed in [126,127].

Pervasive computing systems have a great potential to improve the ways in which we access information. A smart office building would allow employees to perform their jobs better—by changing the nature of computers, employees would be able to concentrate on their work rather than their computers. Sensor networks could allow rescuers to work efficiently and safely and monitor environmental conditions in dangerous areas, allowing residents to evacuate areas before a natural disaster occurs. With this new technology comes a responsibility to maintain the security of the data used in these systems. Security must be considered during the initial design phase and throughout the system's life, ensuring integrity in the face of technology changes and more resourceful attackers. Pervasive computing systems will include devices from a variety of manufacturers, so security protocols must be open standards—allowing any device that meets the system requirements to connect to the system. The future of pervasive computing will depend on the early design of secure systems—users will only accept new technologies if they trust how their information is being used.

Developing security solutions for pervasive systems requires modifying current solutions to take into account the limitations of the mobile and wireless environment. In some cases, developing new solutions will create more eloquent systems. In any case, the lessons learned from the design of traditional and multidatabase systems will be useful.

5. Conclusions

Security is an essential component of any computer system. In this chapter, our goal was to outline the issues and some of the solutions to the problem of security in database systems. We began by focusing on centralized database systems. In this environment, authentication, authorization, and auditing perform conjoint functions and form a powerful weapon addressing database security threats. Authentication is the first gate preventing attackers from penetrating the system. Common authentication techniques take one or a combination of the four basic forms: knowledge-based, token-based, biometric-based, and recognition-based authentications. Once a user's identity is proved, he/she is allowed to access the system resources under the guidance of a reference monitor, which defines the subject–privilege–object relationship. An access matrix can be used to represent the relationships. To achieve implementation efficiency, access control lists or capability lists are usually implemented instead of an access matrix. Access matrices, access control lists, and capability lists provide the mechanisms for access control. Access policies use these mechanisms to enforce security requirements. They can be classified as discretionary, mandatory, and role-based policies. Generally speaking, there is no "best" policy. The appropriateness of a policy depends on the system security requirements. During the authentication and authorization processes, important user and system activities are recorded in audit logs for real-time or posteriori analysis. A well-designed audit mechanism can protect a system by detecting security violations, assisting the discovery of attempted attacks, and examine possible system security flaws.

As database applications grow larger and more complex, geographically distributed heterogeneous databases are required to cooperate and function as one system. A multidatabase system is a widely accepted database organization in this environment. In such systems, security issues become more difficult. Existing security techniques must be extended to handle insecure communication channels and database heterogeneity. Authentication mechanisms developed for centralized system are augmented with countermeasures to deal with insecure communication links. More specifically, cryptosystem based challenge-response, router-based, and agent & model based authentications are commonly used in distributed database systems. In Section 3, we use the Kerberos and SPX systems to exemplify the basic concepts of distributed authentication. We also discuss that different system designs may apply direct, indirect, or global authentication policies. Authorizations in multidatabase systems must cope with the database heterogeneity problem. Many current authorization models derive access control policies at the global level from local policies using one of the three methods: independent approach, top-down derivation, and bottom-up derivation. In Section 3, we also review several authorization models in real systems.

When multidatabases are extended to mobile environments, many new factors must be considered as discussed in Section 4. Mobility adds the challenge of locating users within the service region and creates the possibility of inconsistent network conditions because users are no longer restricted by location. Wireless communication links are inherently insecure and steps should be taken to create secure wireless links. Of course, the low bandwidth of the medium becomes an even larger issue since security is expensive in terms of communication. Further, mobile devices are resource limited with respect to power, processing, storage, and memory capabilities. Not surprisingly, security has high costs in each of these areas. A distinction between information and meta-data can be made, where information is the actual data requested by a user and meta-data could include a user's profile or location information. To ensure user privacy and a secure system, both must be protected.

In mobile database access environments, the conventional client-server programming paradigm exhibits many deficiencies. For instance, it cannot handle the intermittent network connectivity problem adequately. Researchers proposed a new distributed system design paradigm, agent-based programming, as the remedy for the challenges imposed by wireless media. Many research projects have demonstrated the great potential of mobile agents' application in information retrieval systems [129,130]. One must note that incorporating mobile agents into mobile data access systems introduces not only many advantages, but also new security challenges. The types of potential attacks are often categorized as follows: damage, denial of service, breach of privacy, harassment, and social engineering. These attacks may be launched by mobile agents against hosts, by hosts against mobile agents, and among mobile agents. Thus, security techniques must be developed to protect them from each other. Based on the information fortress model [131], the following techniques are designed to protect hosts [132]: authenticating credentials, access-level monitoring and control, code verification, limitation techniques, and audit logging. Mobile agent protection techniques can be categorized into two groups: fault tolerance based and encryption based techniques [132]. Techniques based on fault tolerance aim to make mobile agents robust in unpredictable environment, thus protecting them from malfunctioning hosts, intermittent network connectivity, etc. Techniques based on encryption hide the mobile code or sensitive information so that it cannot be recognized and thus will be less likely to be stolen or misused. Fault-tolerant based techniques include replication and voting, persistence, and redirection. Encryption based techniques include sliding encryption, trail obscuring, code obfuscation, encrypted data manipulation, and state appraisal functions. Mobile agent protection techniques are still in their infancy [132]. A well-defined agent protection model is still missing. To make matters worse, information-fortress-model-based host protection techniques are in direct conflict with many mobile agent protection techniques. Once a mobile

agent is within a host it cannot be protected from destruction, denial of service, etc. Therefore, much research is expected to be done in this area.

Designing and maintaining secure database systems will continue to be a challenge in the future. As mobile computing becomes widely deployed, system designers must extend existing security solutions and develop new solutions that better adapt to the constraints of the mobile environment.

ACKNOWLEDGEMENT

This work in part has been supported by The Office of the Naval Support under the contract N00014-02-1-0282 and National Science Foundation under the contract IIS-0324835.

REFERENCES

[1] Sandhu R., Samarati P., "Authentication, access control, and intrusion detection", in: *The Computer Science and Engineering Handbook*, 1997.

[2] Kohl J.T., Neuman B.C., Tso T.Y., "The evaluation of the Kerberos authentication service", in: Brazier F., Johansen D. (Eds.), *Distributed Open Systems*, IEEE Comput. Soc., Los Alamitos, CA, 1994.

[3] "Data encryption standard", National Bureau of Standards, US Department of Commerce, Federal Information Processing Standards Publication 46, Washington, DC, 1977.

[4] Rivest R.L., Shamir A., Adleman L., "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM* **21** (2) (February 1978) 120–126. See also US Patent 4 405 829.

[5] Yan J., "A note on proactive password checking", in: *ACM New Security Paradigms Workshop, New Mexico, USA*, September 2001.

[6] Klein D.V., "Foiling the cracker: A survey of, and improvements to, password security", in: *Proceedings of the USENIX Security Workshop II*, August 1990, pp. 5–14.

[7] Morris R., Thompson K., "Password security: A case history", *Communications of the ACM* **22** (11) (November 1979) 594–597.

[8] Jobusch D.L., Oldehoeft A.E., "A survey of password mechanisms: weaknesses and potential improvements, part 2", *Computers & Security* **8** (8) (1989) 675–689.

[9] National Computer Security Center, "Password management guideline", Technical Report CSC-STD-002-85, US Department of Defense, April 1985.

[10] Spafford E.H., "OPUS: Preventing weak password choices", *Computers & Security* **11** (3) (1992) 273–278.

[11] Denning P.J., "Passwords", *American Scientist—the Science of Computing* (March–April 1992).

[12] Ganesan R., Davies C., "A new attack on random pronounceable password generators", in: *17th NIST-NCSC National Computer Security Conference*, 1994, pp. 184–197.
[13] Bishop M., "An application of a fast data encryption standard implementation", *Computing Systems* **1** (3) (1988) 221–254.
[14] Farmer D., Spafford E.H., "The COPS security checker system", in: *Proceedings of the Summer Usenix Conference*, Usenix Association, June 1990.
[15] Bloom B., "Space/time trade-offs in hash coding with allowable errors", *Communications of the ACM* **13** (7) (July 1970) 422–426.
[16] Davies C., Ganesan R., "Bapasswd: A new proactive password checker", in: *16th NIST-NCSC National Computer Security Conference*, 1993, pp. 1–15.
[17] Bergadano F., Crispo B., Ruffo G., "Proactive password checking with decision trees", in: *Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, Switzerland*, April 1997, pp. 67–77.
[18] Bergadano F., Crispo B., Ruffo G., "High dictionary compression for proactive password checking", *ACM Transactions on Information and System Security* **1** (1) (November 1998) 3–25.
[19] Husemann D., "The smart card: don't leave home without it", *IEEE Concurrency* **7** (2) (April–June 1999) 24–27.
[20] Shelfer K., "The intersection of knowledge management and competitive intelligence: Smart cards and electronic commerce", in: *Knowledge Management for the Information Professional*, Information Today, Inc., Medford, NJ, 1999.
[21] Flohr U., "The smart card invasion", *Byte* **23** (1) (January 1998) 76.
[22] Shelfer K.M., Procaccino J.D., "Smart card evolution", *Communications of the ACM* **45** (7) (July 2002).
[23] Fletcher P., "Europe holds a winning hand with smart cards", *Electronic Design* **47** (1) (January 11, 1999) 106.
[24] Berinato S., "Smart cards: The intelligent way to security", *Network Computing* **9** (9) (May 15, 1998) 168.
[25] Jain A., Bolle R., Pankanti S. (Eds.), *Biometrics: Personal Identification in Networked Society*, Kluwer Academic, Boston, MA, 1999.
[26] Woodward J., "Biometrics: Privacy's foe or privacy's friend?", *Proceedings of the IEEE* **85** (9) (September 1997).
[27] Braghin C., "Biometric authentication", in: *Research Seminar on Security in Distributed Systems*, Department of Computer Science, University of Helsinki, 2001, http://www.cs.helsinki.fi/u/asokan/distsec/.
[28] Pentland A., Choudhury T., "Face recognition for smart environments", *IEEE Computer* (February 2000), special issue.
[29] Negin M., Camus T., "An iris biometric system for public and personal use", *IEEE Computer* (February 2000), special issue.
[30] Boll S., "Biometrics and the future of money", http://www.house.gov/banking/52098dsb.htm, May 20, 1998.
[31] Shen W., Surette M., Khanna R., "Evaluation of automated biometrics-based identification and verification systems", *Proceedings of the IEEE* **85** (9) (September 1997).

[32] Dugelay J.L., Junqua J.C., Kotropoulos C., Kuhn R., Perronnin F., Pitas L., "Recent advances in biometric person authentication", in: *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, 2002, pp. 4060–4063.
[33] Bleumer G., "Biometric yet privacy protecting person authentication", AT&T Labs-Research, January 1998.
[34] Sims D., "Biometric recognition: our hands, eyes, and faces give us away", *IEEE Computer Graphics and Applications* (1994).
[35] Hays R.J., "INS passenger accelerated service system (INSPASS)", http://www.biometrics.org/REPORTS/INSPASS.html.
[36] Meehan M., "Iris scans take off at airports", July 17, 2000, http://www.computerworld.com/.
[37] Nielsen J., *Usability Engineering*, Academic Press, 1993.
[38] Haber R.N., "How we remember what we see", *Scientific American* **222** (5) (May 1970) 104–112.
[39] Standing L., Conezio J., Haber R.N., "Perception and memory for pictures: Single-trial learning of 2500 visual stimuli", *Psychonomic Science* **19** (2) (1970) 73–74.
[40] Blonder G., United States Patent 5 559 961.
[41] Jermyn I., Mayer A., Monrose F., Reiter M.K., Rubin A.D., "The design and analysis of graphical passwords", in: *Proceedings of the 8th USENIX Security Symposium*, August 1999.
[42] "IDArts", http://www.id-arts.com/technology/papers/, 1999.
[43] Dhamija R., Perrig A., "Déjà vu: A user study. Using images for authentication", in: *Proceedings of the 9th USENIX Security Symposium, Denver, Colorado*, August 2000.
[44] Bauer A., "Gallery of random art", http://andrej.com/art/, 1998.
[45] Bertino E., Samarati P., Jajodia S., "Authorizations in relational database management systems", in: *1st ACM Conference on Computer and Communications Security, Fairfax, VA*, November 3–5, 1993, pp. 130–139.
[46] Bell D.E., LaPadula L.J., "Secure computer systems: Mathematical foundations and model", M74-244, Mitre Corporation, Bedford, MA, 1975 (also available through National Technical Information Service, Springfield, VA, NTIS AD771543).
[47] Sandhu R.S., "Lattice-based access control models", *IEEE Computer* **26** (1) (November 1993) 9–19.
[48] Biba K.J., "Integrity considerations for secure computer systems", Mitre TR-3153, Mitre Corporation, Bedford, MA, 1977. Also available through National Technical Information Service, Springfield, VA, NTIS AD-A039324.
[49] Sandhu R., Ferraiolo D., Kuhn R., "The NIST model for role-based access control: towards a unified standard", in: *Proceedings of 5th ACM Workshop on Role Based Access Control, Berlin, Germany*, 2000.
[50] Ferraiolo D., Kuhn R., "Role-based access controls", in: *15th NIST-NCSC National Computer Security Conference, Baltimore, MD*, October 13–16, 1992, pp. 554–563.
[51] Ferraiolo D., Cugini J., Kuhn R., "Role-based access control (RBAC): features and motivations", in: *Proceedings of the 11th Annual Computer Security Application Conference, New Orleans, LA*, December 1995, pp. 241–248.

[52] Ferraiolo D.F., Barkely J.F., Kuhn D.R., "A role based access control model and reference implementation within a corporate intranet", *ACM Transactions on Information and System Security* **2** (1) (February 1999).
[53] Guiri L., "A new model for role-based access control", in: *Proceedings of the 11th Annual Computer Security Application Conference, New Orleans, LA*, December 1995, pp. 249–255.
[54] Nyanchama M., Osborn S., "The role graph model and conflict of interest", *ACM Transactions on Information and System Security* **2** (1) (February 1999).
[55] Ramaswamy C., Sandhu R., "Role-based access control features in commercial database management systems", in: *Proceedings of the 21st NIST-NCSC National Information Systems Security Conference, Arlington, VA*, October 1998, pp. 503–511.
[56] Sandhu R., Coyne E., Feinstein H., Youman C., "Role-based access control models", *IEEE Computer* **29** (2) (February 1996) 38–47.
[57] Sandhu R., "Role activation hierarchies", in: *Proceedings of the 3rd ACM Workshop on Role-Based Access Control, Fairfax, VA*, October 1998, pp. 33–40.
[58] Sandhu R., "Role-based access control", in: Zelkowitz (Ed.), *Advances in Computers*, vol. 46, Academic Press, San Diego, 1998.
[59] Ting T.C., Demurjian S.A., Hu M.Y., "Requirements, capabilities, and functionalities of user-role based security for an object-oriented design model", in: Landwehr C.E., Jajodia S. (Eds.), *Database Security V: Status and Prospects*, North-Holland, Amsterdam, 1992.
[60] Ferraiolo D.F., Gilbert D.M., Lynch N., "An examination of federal and commercial access control policy needs", in: *NIST-NCSC National Computer Security Conference, Baltimore, MD*, September 20–23, 1993, pp. 107–116.
[61] Garvey T.D., Lunt T., "Model-based intrusion detection", in: *Proceedings of the 14th National Computer Security Conference, Washington, DC*, October 1991, pp. 372–385.
[62] Ilgun K., Kemmerer R.A., Porras P.A., "State transition analysis: A rule-based intrusion detection approach", *IEEE Transaction on Software Engineering* **21** (3) (March 1995) 222–232.
[63] Litwin W., Shan M.C., *Introduction to Interoperable Multidatabase Systems*, Prentice Hall, New York, 1994.
[64] Dawson S., Qian S., Samarati P., "Providing security and interoperation of heterogeneous systems", *Distributed and Parallel Databases* **8** (1) (January 2000) 119–145.
[65] Hurson A.R., Bright M.W., "Multidatabase systems: An advanced concept in handling distributed data", in: *Advances in Computers*, vol. 32, 1991.
[66] Wang C., Spooner D.L., "Access control in a heterogeneous distributed database management system", in: *SRDS*, 1987.
[67] Hammer M., McLeod D., "On database management system architecture", Tech. Rep. MIT/LCS/TM-141, Massachusetts Institute of Technology, Cambridge, MA, 1979.
[68] Heimbigner D., McLeod D., "A federated architecture for information management", *ACM Transactions on Information Systems* **3** (3) (July 1985) 253–278.
[69] Sheth A.P., Larson J.A., "Federated database systems for managing distributed heterogeneous, and autonomous databases", *ACM Computing Surveys* **22** (3) (1990) 183–236.

[70] Veijalainen J., Popescu-Zeletin R., "Multidatabase systems in ISO/OSI environment", in: Malagardis N., Williams T. (Eds.), *Standards in Information Technology and Industrial Control*, North-Holland, The Netherlands, 1988, pp. 83–97.

[71] Vimercati S.D.C.D., Samarati P., "Access control in federated systems", in: *Proceedings of the 1996 Workshop on New Security Paradigms*, September 1996, pp. 87–99.

[72] Alonso R., Barbara D., "Negotiating data access in federated database systems", in: *Proceedings of the 5th International Conference on Data Engineering*, 1989, pp. 56–65.

[73] Hildebrandt E., Saake G., "User authentication in multidatabase systems", in: *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, 1998.

[74] Helal A., Haskell B., Carter J., Brice R., *Anytime, Anywhere Computing, Mobile Computing Concepts and Technology*, in: *The Kluwer International Series in Engineering and Computer Science*, vol. 522, 1999.

[75] Jonscher D., Dittrich K., "An approach for building secure database federations", in: Bocca J.B., Jarke M., Zaniolo C. (Eds.), *Proceedings of the 20th International Conference on VLDB*, Morgan Kaufmann, San Mateo, CA, 1994, pp. 24–35.

[76] Lim J.B., Hurson A.R., "Transaction processing in a mobile, multi-database environment", *Multimedia Tools and Applications* **15** (2) (2001) 161–185.

[77] Laferriere C., Charland R., "Authentication and authorization techniques in distributed systems", in: *Proceedings of IEEE International Carnahan Conference on Security Technology*, 1993, pp. 164–170.

[78] Woo T.Y.C., Lam S.S., "Authentication for distributed systems", *Computer* **25** (1) (January 1992) 39–52.

[79] Tanenbaum A.S., van Steen M., *Distributed Systems: Principles and Paradigms*, Prentice Hall, New York, 2002.

[80] Needham R.M., Schroeder M.D., "Using encryption for authentication in large networks of computers", *Communications of the ACM* **21** (12) (December 1978) 993–999.

[81] Miller S.P., Neuman B.C., Schiller J.I., Saltzer J.H., "Kerberos authentication and authorization system", Project Athena technical plan, section E2.1, October 27, 1988.

[82] Steiner J.G., Neuman B.C., Schiller J.I., "Kerberos: An authentication service for open network systems", Project Athena, March 30, 1988.

[83] Garfinkel S., Spafford G., "Kerberos and secure RPC", in: *Practical Unix Security*, O'Reilly and Associates, Inc., April 1992, pp. 275–291, Chapter 13.

[84] Bryant W., Steiner J., "Kerberos installation notes", Project Athena, MIT, 1989.

[85] Kohl J.T., "Digital Equipment Corporation/Project Athena, Kerberos version 5", presentation material.

[86] Krajewski Jr. M., "Concept for a smart card Kerberos", in: *Proceedings of the 15th National Computer Security Conference*, October 13–26, 1992, pp. 76–83.

[87] Tardo J.J., Alagappan K., "SPX: Global authentication using public key certificates", in: *Proceedings of 12th IEEE Symposium on Research in Security and Privacy, Oakland, CA*, May 1991, pp. 232–244.

[88] Gasser M., Goldstein A., Kaufman C., Lampson B.W., "The digital distributed system security architecture", in: *Proceedings of 12th National Computer Security Conference, Baltimore, MA*, October 1989, pp. 309–319.

[89] Hurton M., "Identification and authentication in distributed systems", in: *IEEE International Conference on Trends in Communications*, July 2001, pp. 394–397.

[90] Templeton M., Lund E., Ward P., "Pragmatics of access control in Mermaid", in: *IEEE-CS TC Data Engineering*, September 1987, pp. 33–38.

[91] Jonscher D., Dittrich K.R., "Argos—A configurable access control subsystem which can propagate access rights", in: *Proceedings of 9th IFIP Working Conference on Database Security, Rensselaerville, NY*, August 1995.

[92] Castano S., "An approach to deriving global authorizations in federated database system", in: *Proceedings of IFIP WG11.3 Working Conference on Database Security, Como, Italy*, July 1996.

[93] Bertino E., Jajodia S., Samarati P., "Supporting multiple access control policies in database systems", in: *Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA*, May 1996.

[94] Wang C.Y., Spooner D.L., "Access control in a heterogeneous distributed database management system", in: *IEEE 6th Symposium on Reliability in Distributed Software and Database Systems, Williamsburg*, 1987, pp. 84–92.

[95] Blaustein B.T., McCollum C.D., Rosenthal A., Simit K.P., "Autonomy and confidentiality: Secure federated data management", in: *Proceedings of the 2nd International Workshop on Next Generation Information Technologies and Systems, Naharia, Israel*, June 1995.

[96] Ngamsuriyaroj S., Hurson A.R., Keefe T.F., "Authorization model for summary schema model", in: *IDEAS*, 2002, pp. 182–191.

[97] Candan K.S., Jajodia S., Subrahmanian V.S., "Secure mediated databases", in: *ICDE*, 1996, pp. 28–37.

[98] Türker C., "An approach to supporting semantic integrity of federated databases", in: *Heterogeneous Information Management, Prague, Czech Republic, 4–5 November 1996, Proc. of the 10th ERCIM Database Research Group Workshop, ERCIM-96-W003*, European Research Consortium for Informatics and Mathematics, 1996, pp. 51–60.

[99] Satyanarayanan M., "Fundamental challenges in mobile computing", in: *PODC*, 1996, pp. 1–7.

[100] Jing J., Helal A., "Client-server computing in mobile environments", *ACM Computing Surveys* **31** (2) (1999) 117–157.

[101] Imielinski T., Badrinath B.R., "Mobile wireless computing: challenges in data management", *Communications of the ACM* **37** (10) (1994) 18–28.

[102] Mouly M., Pautet M.B., *The GSM System for Mobile Communications*, 1992, Palaiseau, France.

[103] Boonsiriwattanakul S., Hurson A.R., Vijaykrishnan N., Chehadeh Y.C., "Energy-efficient indexing on parallel air channel in a mobile database access system", in: *World Multiconference on Systemics, Cybernetics and Informatics: Communication Systems,*

Internet and Mobile/Wireless Computing (SCI'99/ISAS'99), Orlando, FL, vol. 4, 1999, pp. 30–38.

[104] Munoz-Avila A., Hurson A.R., "Energy-efficient object retrieval on indexed broadcast parallel channels", in: *International Conference on Information Resource Management*, 2003, pp. 190–194.

[105] Juran J., Hurson A.R., Vijaykrishnan N., "Data organization and retrieval on parallel air channels: Performance and energy issues", *ACM Journal of WINET*, 2004, in press.

[106] Juran J., "Analyzing conflicts and retrieving requested objects on parallel broadcast channels", Thesis, University Park, PA, 2000.

[107] Bharghavan V., Ramamoorthy C.V., "Security issues in mobile communications", in: *Proceedings of the 2nd International Symposium on Autonomous Decentralized Systems*, 1995, pp. 19–24.

[108] Lubinski A., "Security issues in mobile database access", in: *Proceedings of the IFIP WG 11.3 12th International Conference on Database Security*, 1998.

[109] Hämmäinen H., "Overview of wireless IP devices", in: *Internet Architecture Board, Wireless Workshop, Mountain View, USA*, February 29, 2000.

[110] Haridas H.S., "Security aspects of wireless multidatabases", MS Thesis, The Pennsylvania State University, May 2003.

[111] Solomon J.D., *Mobile IP: The Internet Unplugged*, first ed., Prentice Hall PTR, January 15, 1998.

[112] "Diffie–Hellman key exchange protocol", http://modular.fas.harvard.edu/edu/Fall2001/124/lectures/lecture8/html/node2.html.

[113] "AES Encryption", http://csrc.nist.gov/encryption/aes.

[114] http://csrc.nist.gov/encryption/tkhash.html.

[115] Weiser M., "The computer for the 21st century", in: *Scientific American*, September 1991.

[116] Kahn J.M., Katz R.H., Pister K.S.J., "Next century challenges: Mobile networking for "smart dust" ", in: *ACM Mobicom*, 1999, pp. 271–278.

[117] Satyanarayanan M., "Pervasive computing: Vision and challenges", in: *IEEE Personal Communications*, August 2001.

[118] Schwiebert L., Sandeep K.S., Weinmann J., "Research challenges in wireless networks of biomedical sensors", in: *ACM SIGMOBILE*, 2001, pp. 151–165.

[119] Wood A.D., Stankovic J.A., "Denial of service in sensor networks", in: *Computer*, October 2002, pp. 54–62.

[120] Garlan D., Siewiorek D.P., Smailagic A., Steenkiste P., "Project aura: Toward distraction-free pervasive computing", in: *Pervasive Computing*, April–June 2002, pp. 22–31.

[121] Borisov N., Goldber I., Wagner D., "Intercepting mobile communications: The insecurity of 802.11", in: *ACM SIGMOBILE*, 2001, pp. 180–188.

[122] Perrig A., Szewczyk R., Wen V., Culler D., Tygar J.D., "SPINS: Security protocols for sensor networks", in: *ACM SIGMOBILE*, 2001, pp. 189–199.

[123] Stanford V., "Pervasive health care applications face tough security challenges", in: *Pervasive Computing*, April–June 2002, pp. 8–12.

[124] http://www.privacyrights.org.
[125] Schmidt M., "Subscriptionless mobile networking: Anonymity and privacy aspects within personal area networks", in: *IEEE Wireless Communications and Networking Conference*, 2002, pp. 869–875.
[126] Jiang X., Landay J.A., "Modeling privacy control in context-aware systems", in: *Pervasive Computing*, July–September 2002.
[127] Al-Muhtadi A.R., Campbell R., Mickunas M.D., "A flexible, privacy-preserving authentication framework for ubiquitous computing environments", in: *IEEE International Conference on Distributed Computing Systems Workshops*, 2002.
[128] Lin Y.B., "Mobility management for cellular telephony networks", *IEEE Parallel and Distributed Technology* **4** (4) (1996) 65–73.
[129] Jiao Y., Hurson A.R., "Mobile agents in mobile data access systems", in: *Proceedings of the 10th International Conference on Cooperative Information Systems (COOPIS)*, November 2002.
[130] Papastavrou S., Samaras G., Pitoura E., "Mobile agents for World Wide Web distributed database access", *Transaction on Knowledge and Data Engineering* **12** (5) (September/October 2000).
[131] Blakley B., "The emperor's old armor", in: *Proceedings of New Security Paradigms Workshop*, 1996.
[132] Greenberg M.S., Byington L.C., Harper D.G., "Mobile agents and security", *IEEE Communications Magazine* **36** (7) (July 1998) 76–85.

Disruptive Technologies and Their Affect on Global Telecommunications

STAN MCCLELLAN

Hewlett Packard Carrier-Grade Systems
6900 Dallas Parkway, Plano, TX, USA
stan.mcclellan@hp.com

STEPHEN LOW

Hewlett Packard Carrier-Grade Systems
14231 Tandem Blvd, Austin, TX, USA
stephen.low@hp.com

WAI-TIAN TAN

Hewlett Packard Mobile & Media Systems Lab
1502 Page Mill Rd, Palo Alto, CA, USA
wai-tian_tan@hp.com

Abstract

This chapter examines key architectural, marketectural, and technological issues surrounding "disruptive technologies" which are affecting the structure and operation of global telecommunications. We attempt to categorize and highlight the several key technologies, platforms, and services which are affecting the evolution of telecommunications networks. Many of these functional elements have been driven by consumer demand and a redefinition of "legacy services" in the context of packet-based capabilities (or limitations). In this context, we survey issues related to Quality of Service in packet-switched networks, the commoditization of hardware, operating system, and access bandwidth, and some key protocols affecting the structure and operation of telephony networks.

ISSN: 0065-2458/DOI 10.1016/S0065-2458(03)61004-1
Copyright © 2004 Elsevier Inc.
All rights reserved.

1. Background & Motivation

This chapter examines key architectural, marketectural, and technological issues surrounding "disruptive technologies" which are affecting the structure and operation of global telecommunications. The trends and technologies discussed here are identified as "disruptive" because they are representative of the drastic changes occurring in the field of telecommunications, and they present very different value propositions than their forbears [1]. As such, they are key contributing factors to the convergence of telecommunications, computer, and networking technologies.

Some of the most prominent factors affecting telecommunications are related to the commoditization of hardware, software, and bandwidth. Common among these factors is the presence and influence of the global Internet as a tool for business, entertainment, and information. As a result of these factors, telecommunications revenue (average revenue per user, ARPU) is steadily declining and data services are dominating the bandwidth of networks engineered for voice. To compound these issues, wireless connectivity for voice and data is proliferating at astounding rates. The following sections discuss some of the technologies and issues that are disrupting the status quo and driving this "convergence revolution."

As the primary disruptive force, the effect of the Internet and packet-switched data are discussed in Section 2. This discussion centers on the definition of "Qual-

ity of Service" and technologies enabling packet-switched networks to adapt to the requirements of multiservice dataflows. In concert with Internet proliferation, the commoditization of bandwidth and network access is discussed in Section 3, highly-available and high-performance operating systems are examined in Section 4, and powerful, low-cost computing elements are considered in Section 5. In a potent combination of price, performance, and ubiquity, these factors are lowering traditional barriers for entry into telecommunications, and fanning the flames of the information (r)evolution. To complete the overview of significant disruptors, Section 6 discusses a few of the user-level and transport interfaces that are affecting the fundamental structure and operation of communications networks. Finally, in Section 7 we conclude with the observation that a fundamental premise of converged networking is the migration of enhanced services from centralized facilities toward the open and distributed "intelligent edge."

2. Quality of Service: Packet vs. Circuit

The convergence of technology, capabilities, business models, and user expectations via Internet connectivity has greatly challenged the communications infrastructure, and "disrupted" the status quo in many ways. As high-performance, low-cost, communications technology is deployed in consumer-grade devices, the fundamental task of communication has been undergoing a drastic paradigm shift. This change is evidenced most clearly and driven most directly by the migration of intelligence (or, processing power) toward the edge of the network [2]. Accordingly, an important issue in the migration from centralized network control to decentralized or distributed network control is the complex matter of guarantees on Quality of Service (QoS).

QoS is a vague, all-encompassing term used to describe an environment where the network provides some type of preferential delivery service or performance guarantees, e.g., guarantees on throughput, maximum loss rates or delay. Network-based support for QoS can greatly facilitate media communication, as it enables a number of capabilities including provisioning for media data, prioritizing delay-sensitive video data relative to other forms of data traffic, and prioritizing among the different forms of media data that must be communicated. Circuit-switched networks, such as the legacy wired and wireless telephony networks, are designed specifically to provide QoS guarantees for transport of a specific class of information (voice). Unfortunately, QoS is currently not widely supported in packet-switched networks such as the Internet.

Specific ways in which streaming media systems can take advantage of a QoS-enabled Internet is currently an area of active research, because performance optimization for complex, IP-based distributed systems is quite difficult. End-to-end

network performance depends on many factors from the digital signal processing techniques used in media compression to the management, integration, and policy enforcement technologies for the overall network. Multimedia streams not only demand higher bandwidth but also have peculiar timing requirements such as lower delay tolerance and higher end-to-end delivery guarantees than other data. In contrast, IP network design is based on best-effort packet forwarding, an approach that doesn't distinguish explicitly between the needs of particular streams. Some of the efficiency and scalability of IP networking is based directly on the fact that intermediate forwarding devices *don't need to consider* explicit per-packet or per-stream requirements. Unfortunately, adequate QoS may require some change to this fundamental paradigm.

The definition of mechanisms to support QoS in IP-based networks has been an important area of network research over the past two decades. The resulting IP QoS framework lets network elements discriminate between particular traffic streams and then treat those streams in a particular manner, subject to broad constraints on forwarding performance.

Several Internet Engineering Task Force groups (IETF) [3] have been working on standardized approaches for IP-based QoS technologies. The IETF approaches of interest here fall primarily into the following categories:

- prioritization using differentiated services (RFC 2475),
- reservation using integrated services (RFC 2210), and
- multiprotocol label switching (RFC 3031).

The Integrated Services model of the Internet (IntServ, RFC 2210) is an attempt to define mechanisms for end-to-end QoS guarantees for bandwidth, packet loss rate, and delay, on a per-flow basis. QoS guarantees are established using explicit resource allocation based on the Resource Reservation Protocol (RSVP). IntServ emulates the resource allocation concept of circuit switching to apportion resources according to requests made by each host. The initiating host sends the upper-bound specification of bandwidth, delay, and jitter, and intervening nodes forward that request until it reaches the receiving host. Two kinds of reservations are defined in the IntServ model. A *guaranteed reservation* prescribes firm bounds on delay and jitter, determined from the network path, and allocates bandwidth as requested by the initiating host. *Controlled-load reservations* depend on network congestion, thus they work only slightly better than best-effort service.

The high complexity and cost of deployment of the IntServ architecture led the IETF to consider other QoS mechanisms. The Differentiated Services model (DiffServ), in particular, is specifically designed to achieve low complexity and easy deployment at the cost of less stringent QoS guarantees than IntServ. Under DiffServ,

service differentiation is no longer provided on a per-flow or per-request basis. Instead, packets are classified into predefined per-hop behaviors based on a *DiffServ code point* (DSCP) or tag attached to the type of service byte in the IP header. The DiffServ *expedited forwarding class* (EF) is intended to provide the highest level of aggregate QoS for streams requiring minimal delay and jitter. Other classes, such as *assured forwarding* (AF) and *best effort* (BE) classes detail coarse loss bounds and relative prioritization between streams. Unfortunately, the transport effectiveness of DiffServ-marked packets depends heavily on the per-hop behaviors implemented in the intervening nodes.

Label switching refers to a type of traffic engineering. With this approach, a router determines the next hop in a packet's path without looking at the header of the packet or referring to routing lookup tables. The IETF's multiprotocol label switching (MPLS) architecture assigns short, fixed-length labels to packets as they enter the network. The network uses these labels to make forwarding decisions, usually without recourse to the original packet headers. The DiffServ model combined with MPLS label-switched paths in the core of the network with some implementation of RSVP in the access networks may be a natural and effective "hybrid" approach to Internet QoS. However, the issues of end-to-end coordination between the specific meaning of traffic classifications, the accurate mapping of these classifications to labels, and the deployment of consistent queuing & forwarding behaviors at intermediate nodes is an extremely complex undertaking.

Table I lists a selection of IP-based QoS mechanisms that pertain primarily to DiffServ and IntServ networks. The table separates these QoS mechanisms into *queuing strategies* or *reservation, allocation, and policing techniques*. Under the proper circumstances, these mechanisms, which are available in conventional packet forwarding systems such as IP routers, can differentiate and appropriately handle isochronous (or time-sensitive) traffic. Following sections discuss some of the key considerations and architectures in the transport of voice and video over IP-based, QoS-enabled networks. These concepts will be increasingly important as IP networking continues its "disruption" of legacy telecommunications infrastructure.

2.1 End-to-End QoS for Voice

Studies of voice quality have demonstrated the relative performance of various QoS implementations under specific but broadly applicable network architectures and performance conditions [4]. These results reinforce the observation that joint optimization of network characteristics in the presence of general application-level traffic is an extremely complex issue. Adequate QoS requires joint optimization of queuing strategies, call-admission controls, congestion-avoidance mechanisms, and

TABLE I
IP QOS MECHANISMS

Description	Acronym	Comment
		Queuing
First In First Out	FIFO	Also known as the Best-Effort (BE) service class, packets are simply forwarded in the order of their arrival.
Priority Queuing	PQ	Allows prioritization on pre-defined criteria or policies. Based on the policies, arriving packets are placed into one of four queues – high, medium, normal, and low priority. DSCP packet marking can be used to prioritize such traffic.
Custom Queuing	CQ	Allows allocating a specific amount of a queue to each class while leaving the rest of the queue to be filled in round-robin fashion. It essentially facilitates prioritizing multiple classes in queuing.
Weighted Fair Queuing	WFQ	A queuing strategy which schedules interactive traffic to the front of the queue to reduce response time, then fairly shares the remaining bandwidth among high-bandwidth flows.
Class-Based Weighted Fair Queuing	CBWFQ	Combines custom queuing and weighted fair queuing. CBWFQ gives higher weight to higher-priority traffic which is defined in classes using WFQ processing.
Low-Latency Queuing	LLQ	Brings strict priority queuing to CBWFQ. It gives delay-sensitive data preferential treatment over other traffic.
		Reservation, allocation, and policing
Resource Reservation Protocol	RSVP	A signaling protocol, provides setup and control to enable the resource reservation that integrated services prescribe. Hosts and routers use RSVP to deliver QoS requests to routers along data stream paths and to maintain router and host state to provide the requested service.
Real-Time Transport Protocol Queuing	RTPQ	RTP queuing offers another way to prioritize media traffic. Media packets usually rely on the user datagram protocol with RTP headers. RTPQ treats a range of UDP ports with strict priority.
Committed Access Rate	CAR	A traffic-policing mechanism, CAR allocates bandwidth to traffic sources and destinations while specifying policies for handling traffic that exceeds the allocation.

traffic-shaping & policing technologies. Different QoS technologies—each addressing different aspects of end-to-end performance and each implemented in particular ways by equipment vendors—must be carefully evaluated to achieve adequate QoS for isochronous data, and in particular for Voice over IP (VoIP) deployments. Unfortunately, many of the popular remedies for packet-based network issues aren't directly applicable to voice-grade transport. Four of the network parameters most

important in the effective transport of multiservice traffic are bandwidth, delay, jitter, and packet loss. Measuring and validating these parameters is difficult, particularly in the context of a highly subjective and variable phenomenon such as voice quality. Thus, VoIP networks may be "optimized" according to ineffective parameters, or over-engineered to ensure sufficient resources. In particular, the lack of a dedicated signaling channel in IP networks is problematic, because failure of call-signaling messages due to network congestion leads to catastrophic failure of the bearer channel. The complex interrelationships between network configuration, packet-forwarding technologies, and effective end-to-end results may require indirect optimization. In these cases, the use of subjective voice quality as a performance metric for network configurations can provide valuable insights.

Formal and informal subjective rankings (Mean Opinion Score, MOS) for voice transmission quality over a DiffServ-enabled IP network are summarized in Fig. 1. In situations where a "bottleneck" network architecture similar to Fig. 4 exists, voice streams can be prioritized using several of the QoS technologies described in Table I. When the channel is oversubscribed with a combination of voice, data and

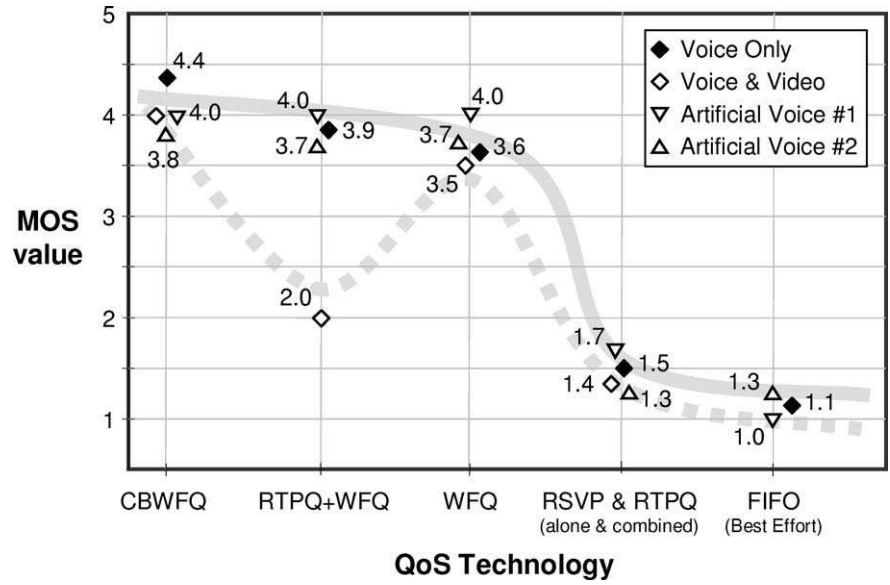


FIG. 1. Subjective (Mean Opinion Score, MOS) rankings for QoS mechanisms in the transport of voice in a congested DiffServ network. Data labeled "Voice" & "Voice & Video" was compiled from a group of listeners. Data labeled "Artificial Voice" was taken from commercially-available voice quality testing products.

video traffic, these prioritization mechanisms can be effective in preserving the subjective quality of the voice streams in addition to maintaining acceptable video fidelity. The clearest observation that can be made with reference to Fig. 1 is that QoS mechanisms which utilize Weighted Fair Queuing (WFQ) tend to control traffic throughput in a more deterministic manner than mechanisms based on reservation (RSVP), port-based treatment (RTPQ), or the default best-effort (FIFO). Additionally, and although Fig. 1 doesn't explicitly reflect this result, a lack of relative differentiation between voice and video in the DiffServ Expedited Forwarding (EF) class results in heavier video packet losses in the RSVP and RTP-based methods (alone or in combination) since high-rate video must be "demoted" to an assured-forwarding (AF) class to preserve voice throughput. Similar mechanisms are at-play in the case of video transport for the WFQ + RTPQ mechanism. However, in this case the lack of differentiation between video and voice (both are RTP streams) results in relatively better-quality transport for video and heavier packet loss for voice streams. As a result, WFQ + RTPQ produces relatively poor subjective quality when video (or other RTP-based streams) are present in the network. These effects can be compensated by implementing QoS schemes with higher effective differentiation between traffic streams, such as Class-Based Weighted Fair Queuing (CBWFQ). In CBWFQ, streams are sorted into classes with independent WFQ implementations, and forwarding mechanisms can be carefully tailored based on classes and queues within classes. Clearly, the fidelity of CBWFQ performance for a network requires detailed insight into traffic characterizations as well as significant real-time processing on packet streams. However, QoS mechanisms which approximate per-stream assurances through classification and careful forwarding techniques tend to perform better than mechanisms that simply reserve bandwidth, ensure low latency for forwarding without classification, or prioritize based on generic traffic classifications.

2.2 Video over DiffServ Networks

There are a number of ways to exploit service differentiation for video communication in a DiffServ-enabled Internet. Given a fixed number of traffic classes, one strategy is to assign different data types to different service classes. For example, one traffic class may be used for video, one for interactive data applications such as web browsing, and one for non-interactive data transfer such as email and file transfer. By providing lowest loss and delay to video followed by interactive data and non-interactive data respectively, the user expectations for the different applications can be better met. However, as mentioned previously, if there are multiple isochronous streams, differentiation between them can be complex and application-dependent. So, despite the fact that data traffic is loss sensitive, it can be transported

using higher loss classes because data traffic is insensitive to delay, and the use of end-to-end IP retransmission protocols such as the Transport Control Protocol (TCP) can effectively translate loss into additional delay and delay-jitter. Another strategy for service differentiation is to reduce the average number of playback pauses by exploiting the behavior of TCP congestion control mechanisms in streaming media over the Hypertext Transport Protocol (HTTP).

2.2.1 Delay and Loss Differentiation in MPEG Video

A refinement of the multi-class strategy mentioned above is to use multiple traffic classes even within a single video application. This technique is implemented by decomposing a video bit-stream into multiple sub-streams of different delay or loss sensitivities, and assigning each sub-stream to a different traffic class with a different QoS characteristic. This "multi-class" approach generally achieves better utilization of network resources than the "single-class" approach due to the ability to transmit packets using traffic classes with commensurate QoS. Another advantage of the multi-class approach is the fine granularity tradeoff between network resources and reception quality. This results in a spectrum of cost-performance tradeoff in transmitting video over DiffServ networks.

The idea of providing differential treatment to different parts of compressed digital video is not new. In particular, separating the source material into a base layer and an enhancement layer and applying priority queuing or different loss probabilities to the layers improves performance for statistically multiplexed networks [5–7]. Two-layer scalable MPEG also outperforms single-rate MPEG with loss prioritization [8,9]. For existing content compressed using one-layer MPEG, different priorities can be assigned according to the different I, P, and B frame types, or the motion information can be prioritized differently than the texture information [10, 11]. An alternate approach to exploiting QoS diversity is to transcode the MPEG video into formats that can be easily mapped to different network traffic classes. For instance, transcoding video into a multi-layered representation would allow natural mapping of layers to traffic classes with increasing loss rates. Unfortunately, transcoding suffers from implementation issues, including video quality degradation, high complexity, and increased latency. These issues can be problematic for real-time transmissions.

A transcoding-free approach to sending video with multiple traffic classes in a DiffServ network entails splitting an MPEG bit-stream into sub-streams having different delay and loss sensitivities [12]. Via a preprocessing technique, a map of loss-delay spread is obtained showing the amount of video data with each value of delay and loss sensitivity, as illustrated in Fig. 2(a). Since the map orders different parts of an MPEG video according to delay and loss sensitivities, it can be used for packet

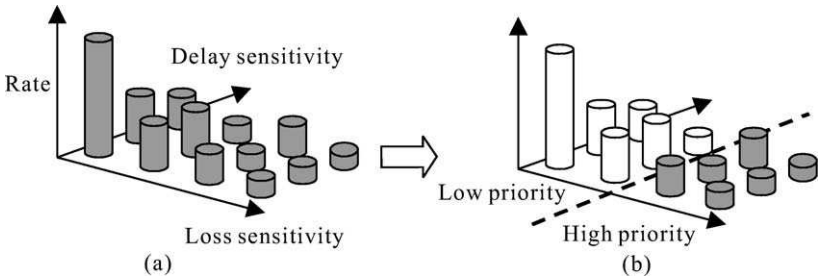


FIG. 2. Using a map of delay-loss spread to classify traffic into low-loss and high-loss classes.

classification and tagging in delay and loss differentiated networks. For example, Fig. 2(b) illustrates the classification of an MPEG video bit-stream into two traffic classes with different loss priorities. Such preprocessing techniques are compatible with the large body of existing, single-layer MPEG sources, and so have wide applicability. The loss-delay map is also a property of the video content and format, and so can be computed independently of the network traffic classes prior to transmission.

Structured transmissions which prioritize motion information over texture information [11] or assign different priorities to I, P, and B frames [10,13] are effective in exploiting the loss-sensitivity of MPEG video. These techniques lead to a variety of algorithms, described in Table II, which produce loss-sensitivity metrics for video streams. These algorithms can be ranked in terms of effectiveness for transmitting video over a DiffServ network. In this context, the technique of relative frame priorities can be generalized using the notion of *reference counts*, in which each video frame or portion of a frame ("slice") is assigned importance according to how many frames or slices are dependent on it, and whether *motion* information is differentiated from *texture* information.

The relative performance characteristics of these techniques can be compared in the context of a simple loss-differentiated network. Figure 3 shows simulation results for such a scenario where the network has two traffic classes: low-loss (0.3% loss rate) and high-loss (3% loss rate). In Fig. 3, 30% of the available bandwidth is allocated to the low-loss class and distortion is measured in mean square error (MSE) as a function of time. The MSE induced by packet loss is computed for every video frame, and an average number is reported for every minute (1800 frames). In the case when a slice is missing due to packet loss, it is approximated either by the corresponding slice in the previous reconstructed frame, or the previous slice in the current frame, depending on which is estimated to be more similar. From the figure, it is clear that the *Uniform* scheme, which offers no loss

TABLE II
MECHANISMS FOR DIFFERENTIATED TRANSMISSION OF DIGITAL VIDEO

Technique	Description
	Texture-based differentiation
Uniform	Treats an MPEG bit-stream as homogeneous and assigns equal importance to all bits. Doesn't differentiate between *motion* information and *texture* information.
Frame-Level Reference	All slices in frame i have equal importance determined by the number of other frames that refer to frame i. For example, every B frame has a reference count of one since no other frames besides itself refers to it. On the other hand, an I frame followed by 4 P frames and 8 B frames will have a reference count of 13.
Slice-Level Reference	Slices in frame i have unequal importance determined by their use in predicting subsequent frames. For example, if K 16×16 macroblocks in frame i are used in predicting a macroblock in subsequent frame j, each of the K macroblocks in frame i will receive a contribution of $1/K$ to their reference count. The importance of each slice in frame i is then computed as the average reference count of its macroblocks.
	Motion & Texture-based differentiation
Motion	Assigns more importance to motion information than texture.
Motion + Slice-Level Reference	Motion information is assigned the highest importance, and the importance of texture information is determined via *Slice Level Reference*. Differentiates explicitly between *motion* and *texture* information.

differentiation, suffers from the highest distortion because all stream segments are treated in identical fashion. Additionally, techniques which incorporate *Slice Level Reference* to increase loss differentiation consistently outperform other schemes, but at the expense of increased complexity. However, note that the *Motion + Slice Level Reference* technique produces lower average distortion than any of the other schemes, because loss differentiation based on a combination of motion and texture information is more effective than differentiation based purely on reference counts.

While MPEG compression does not make special provisions for transport over delay-differentiated networks, limited delay separation is achievable by exploiting the difference between the order in which video frames are stored in the bit-stream, and the order in which they are displayed. This difference between the bit-stream and display order of MPEG video means that certain frames can afford to have larger transmission delays than others. Generally, intra-coded (I) and predicted (P) frames can tolerate more delay (up to a video frame-time) than bi-directionally predicted

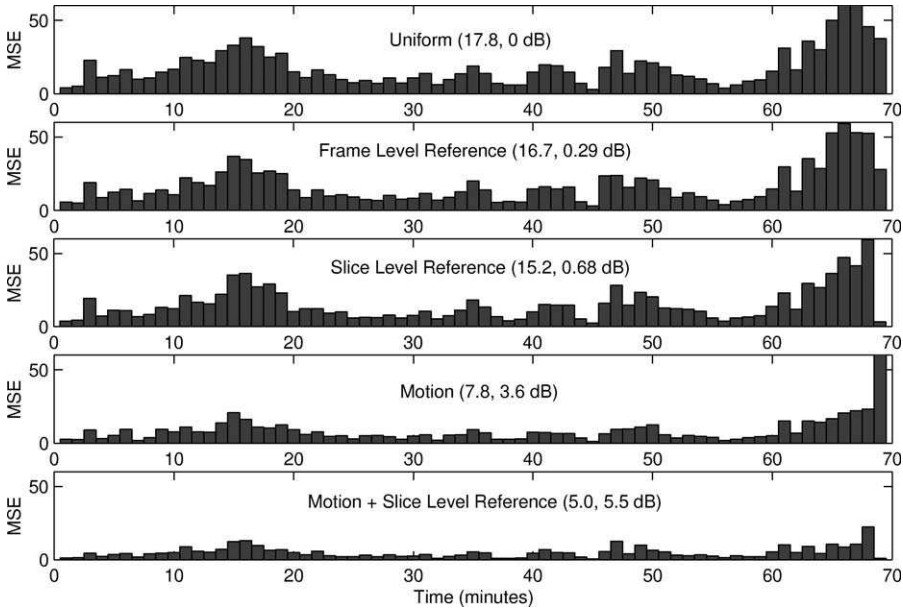


FIG. 3. Comparison of several loss differentiation schemes using a two-class (low-loss, high-loss) network with 30% bandwidth allocated to low-loss traffic. Average MSE and distortion reduction in dB for each scheme are shown in brackets.

(B) frames. For a full frame rate movie at 30 frames/s, this corresponds to a 33 millisecond difference in delay requirements, which can be significant for network transmission. Additionally, implementation complexity is extremely low since delay spread is obtained using frame-level information already encoded in the MPEG picture headers. The delay spread can also be improved by examining the motion vectors to determine the actual dependence in the video.

For networks that offer differentiation in delay as well as loss, the packet tagging algorithm is more challenging. Generally, if the available rates for different traffic classes are known, then a tagging scheme that simultaneously optimizes loss and delay may not exist. It is possible to construct an aggregate cost as a function of loss and delay so that the different traffic classes can be ordered in terms of QoS, and the delay-loss spread map can be ordered in terms of importance. While such an approach would allow simple mapping of data to different traffic classes, the construction of such a cost function requires a trade-off between delay and loss. Since delay and loss give rise to different types of degradations in visual communication, a more natural approach is to optimize first for either loss or delay.

2.2.2 Reducing Playback Pauses in DiffServ Networks

An approach to compensating for delay sensitivity in network video is to couple feedback from application-level buffering mechanisms with the priority and forwarding mechanisms of the network. In streaming media applications, media data has to be continuously available at the receiving node to avoid playback disruption. Buffering is normally used at the receiver to reduce the frequency of playback disruptions, but levels of buffer depletion can vary widely among multiple receivers using the same link. This implies that playback disruption may be controlled by allocating more instantaneous network resources to media sessions experiencing playback buffer depletion. To accomplish this instantaneous allocation, application-level indicators of buffer utilization can be inserted into packet headers, bypassing any requirement for explicit network reservation. For networks that provide two or more service classifications, joint prioritization across all streaming sessions can be performed dynamically at resource bottlenecks using the labels in the packets alone. Since the transport prioritization at the resource bottleneck is based on labels carried in the packet and without any other streaming or transport session information, the approach can be called Playback-Buffer Equalization using Stateless Transport Prioritization (PBE-STP) [14]. Specifically, PBE-STP involves inserting a label in each transmitted packet that corresponds to the buffer occupancy of the streaming session. In this fashion, streaming sessions experiencing playback buffer depletion may be transmitted using a higher grade of service. As a result, the end-to-end congestion control mechanisms of the depleted stream are indirectly triggered to operate at higher throughput at the expense of streaming sessions that already have plenty of buffer. PBE-STP may be beneficial in practical scenarios involving multiple users behind a single resource bottleneck. Advantages of PBE-STP include:

- No client or protocol modification—streaming clients and protocols that are already widely deployed today can be used directly.
- No network modification—no media-specific knowledge is required for the network, so that existing QoS mechanisms such as DiffServ can be employed.
- Multiple sessions may be delivered from multiple servers, and no coordination among the servers is required.
- No frequent setup/teardown or reconfiguration of the QoS network is required since prioritization of resources is across all media sessions sharing the bottleneck resource.

For example, consider the QoS-enabled network of Fig. 4, consisting of *Media Streaming Servers*, *Packet Classifiers*, and *Streaming Clients*. In the figure, the network offers at least two service classifications that differ in end-to-end packet loss rate, delay, or both. The *Media Streaming Server* implements any congestion control

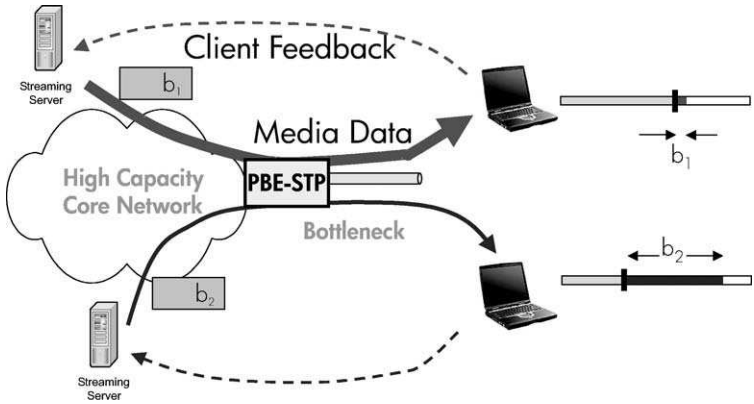


FIG. 4. Architecture of a PBE-STP network with *Media Streaming Servers*, *Packet Classifiers*, and *Streaming Clients*. The network and streaming clients do not require modifications.

algorithm commonly used for streaming media. For each of the streaming sessions, the streaming servers first derive the client buffer occupancy of each session using existing knowledge of the start-time and the amount of data transmitted for that session. A label representing the time-varying Time-To-Depletion for each session is then recorded in each IP packet belonging to that session. Time-To-Depletion for a streaming session is the time for which media playback can be sustained using only data that has already been buffered (playback buffer occupancy). For each media stream, the *Packet Classifier* uses the Time-To-Depletion labels to assign packets to different available service classes. Specifically, packets with labels corresponding to a smaller Time-To-Depletion are preferentially given better service. The *Streaming Clients* are compatible with the associated *Media Streaming Server*, and are unaware of the presence and actions of the *Packet Classifier*. In this scenario, transmitting packets of a particular session using a better service classification has a number of effects in addition to faster delivery or lower loss probability. For instance, the "upgraded" sessions will effectively observe a "better" channel, and by the virtue of congestion control, boost their transmission rates. Typical congestion control mechanisms rely on end-to-end measurable quantities such as round-trip propagation delay and packet loss statistics to infer the level of congestion in the transmission path. When the congestion level is high, the typical action is to lower the transmission rate so that through the collective action of all sessions, the level of congestion can be controlled. So, despite the fact that congestion control mechanisms in the streaming servers and clients are not aware of the existence of PBE-STP, the preferential transport for sessions experiencing playback buffer depletion will trigger higher transmission rates to those sessions with most need. Additionally, since the assignment

of packets to service classes is performed locally at the resource bottleneck, global optimization for all streams across different servers can be achieved.

Figures 5 and 6 show the effect of PBE-STP in an experimental network similar to Fig. 4. In this case, 9 flows each with constant bit-rate 1.2 Mbps are streamed from the *Media Streaming Servers* to the clients through a 10 Mbps bottleneck at the *Packet Classifier*. Six of the flows start at time 0 and the remaining three flows starts at times 60, 120, and 180 seconds, respectively. To effect packet classification, each of the *Media Streaming Servers* records a 3-bit label representing the time-varying Time-To-Depletion for each media session in the Type-Of-Service (TOS) field of each IP packet. Clearly, this approach is compatible with DiffServ implementations if the Time-To-Depletion tag is appropriately related to DiffServ DSCP values and per-hop behaviors. The top graph of Fig. 5 shows the amount of data received by the different streaming sessions in a typical run of the scenario. The amount of data is reported in seconds, and is the ratio of the amount of received data in bits to the bit-rate of the media. Media playback starts five-seconds after streaming starts

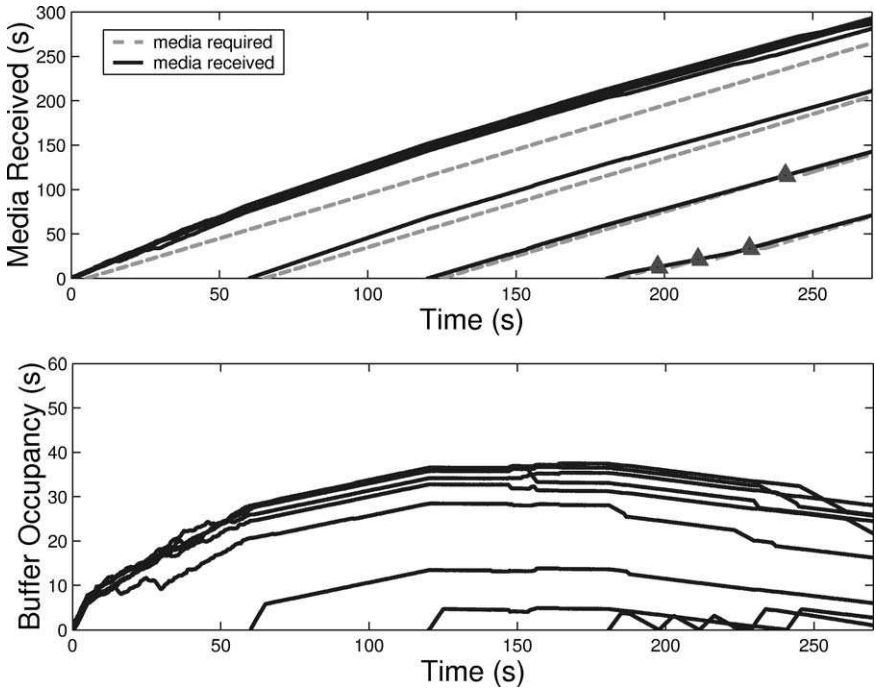


FIG. 5. Without hybrid priority queuing, media sessions with later start-times experience playback buffer depletion, resulting in several playback disruptions.

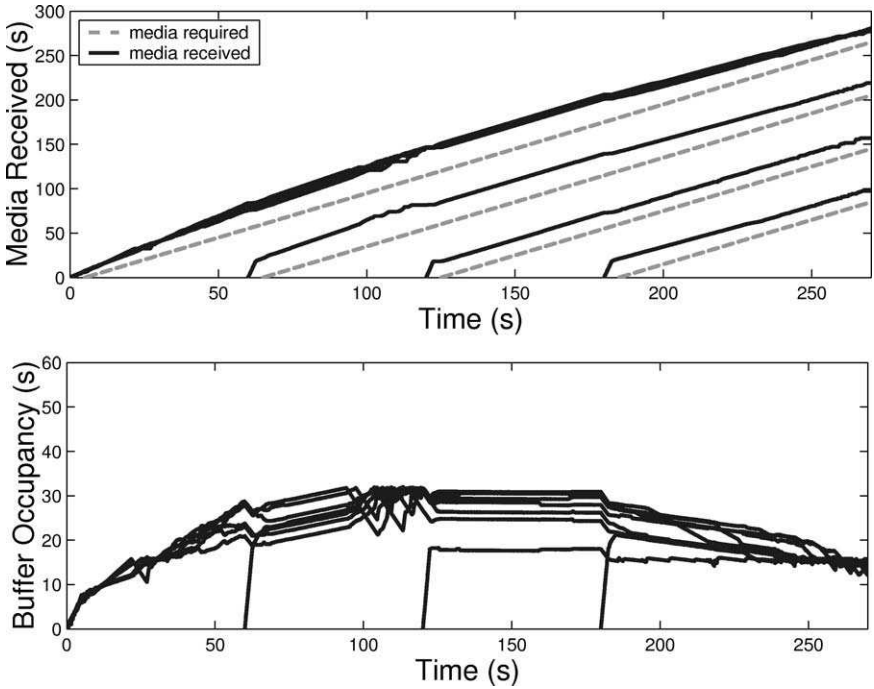


FIG. 6. Using hybrid priority queuing, PBE-STP alleviates discrepancies in client buffer occupancy and minimizes playback disruptions.

(i.e., 5 s pre-roll buffer). The dotted lines show for each stream the amount of data that needs to be consumed as a function of time to maintain the streaming session with no playback disruption. In other words, playback disruption occurs when the solid line crosses the corresponding dotted line, as indicated by triangular markers in Fig. 5. After a playback disruption, the client pauses and rebuffers for 5 seconds before resuming playback. The difference between the solid and the dotted lines of a streaming session is the client buffer occupancy, which is shown in the lower plot of Fig. 5. There is generally a very large discrepancy in the amount of client buffer across sessions that start at different times. Between time 0 to 60 seconds, 6 sessions share the bottleneck link, and all sessions are building up their client buffer. Around 150 seconds, 8 flows are sharing the bottleneck link, and none is gaining in buffer occupancy. Finally, when the last session start at time 180 seconds, the throughput that each of the flows receive is lower than the required media rate, so all clients are draining their playback buffer. The last stream, with the least buffer build-up, is the first to experience playback disruption, followed by the second to

last stream. Furthermore, the last two sessions experience playback disruptions while other sessions have ample amount of buffered data at the client. The goal of PBE-STP is to exploit such discrepancy to provide higher throughput to sessions with less buffer at the expense of sessions with more buffer.

An important criterion in selecting the queuing discipline is to avoid starvation of streams. This is because starvation can trigger congestion control mechanisms (such as the exponential back-off algorithm of TCP), and subsequently prevent streams from resuming even when resources become available. An approach to prevent starvation is to employ hybrid priority queues that switch between round-robin and priority queues in a periodic fashion. Specifically in the context of the PBE-STP network of Fig. 4, Fig. 6 summarizes the resulting playback buffer situation when the *Packet Classifier* implements a hybrid queuing discipline with 8 queues corresponding to 8 possible Time-To-Depletion labels. In this case, packets are served according to two alternating rules: every K packets served according to priority queuing are followed by one packet served according to round-robin. The corresponding results for $K = 15$ are shown in Fig. 6. Compared to Fig. 5, there are fewer playback disruptions, and the discrepancy in client buffer occupancy between streams is improved.

3. Commodity Bandwidth

In many cases, and despite its obvious advantages, IP-based networking has been shown to be lacking in performance, manageability, or market penetration. In many of these cases, the rallying cry of IP afficionados has been "throw more bandwidth at it." The following sections discuss technologies which reveal flaws in the "more bandwidth" argument. In these cases, bandwidth is either already plentiful or it has very limited prospects for increase. These technologies are "disruptive" in the telecommunications space because of many factors, including cost and speed of adoption. For instance, the deployment of secure, robust, and manageable architectures which integrate wireless LAN "hotspots" with existing operator networks has become an important topic for wireless telephony.

3.1 Wired: "Fiber, Fiber Everywhere but not Enough Apps on the Link"

Recent investment in optical fiber infrastructure has been significant, particularly in dense urban areas and along "backbone" data traffic routes. However, the equipment, space, and ongoing maintenance costs for an active network infrastructure far outweigh the one-time investment of laying cable. As a result, much more "dark

fiber" is deployed than may be used in the near future. For instance, by many estimates less than 5% of deployed fiber in the United States is actively carrying traffic. Although excess "dark fiber" doesn't translate directly into excess available bandwidth without additional investment in equipment, the relatively anemic usage of deployed transport facilities is still somewhat puzzling. Based on conventional wisdom (and common sense, e.g., declining bandwidth costs), a reason for the underutilization of these facilities may be that mass-consumer need for broadband connectivity is not (yet) compelling. Combined with the relatively slow rollout and uptake of broadband access networks such as cable modem and Digital Subscriber Loop (DSL), a lack of "killer applications" for bandwidth use has led to an imbalance of supply and demand.

In similar fashion, although wireless voice telephony remains popular, the delayed rollout of broadband wireless services may be a moot point unless consumer demand justifies the up-front capital costs of spectrum licenses and equipment upgrades. Industry uncertainty about the popularity of "3G" applications such as mobile Internet service will be justified unless the benefit to the consumer outweighs the cost and wireless data becomes a "must have" service. Additionally, if handheld devices are too inconvenient to use, or are incapable of a positive "user experience", the available bandwidth—and expense of delivering it—will be wasted.

3.2 Wireless: 3G, 4G, and 802.11

Two information services that have experienced tremendous growth in recent years are on a collision course. As separate entities, wireless telephony and the Internet have affected personal and business interactions in a profound way. One of the major disruption fronts in this realm is the melding of conventional wireless telephony with burgeoning IEEE 802.11 wireless LAN (WLAN).

As a data service, WLAN has many advantages over conventional telephony, including low cost, ease of operation, higher throughput and lower latency. For instance, "2.5G" GPRS currently offers less than 128 kbps, with GPRS/EDGE enhancements promising less than 384 kbps. The effective bandwidth of "3G" rollouts will also be substantially less than 1 Mbps. These rates may be inadequate for the heavy transactions of pseudo-stationary mobile computing applications. So, although wireless telephony has more seamless coverage and mobility support, an 802.11b network that can sustain 5 Mbps throughput with negligible delay [15] will be attractive in some cases. However, 3G radio spectrum is *licensed* and *regulated* so that services can be planned and deployed in a more predictable manner. In contrast, 802.11 operates in *unlicensed* and *unregulated* spectrum where there are no "rules" to prevent or limit interference from a neighbor or competitor using the same spectrum. Without careful management and network architecture, the net result could

be relatively unpredictable performance for WLAN users. Although voice transport will likely continue to be the primary service for wireless telephony networks, multimode WLAN products are emerging that will challenge the throughput and utilization modes of today's 2.5/3G networks and extend their scope to accommodate mobile "power users."

In response to these market dynamics, licensed telephony carriers have begun to complement their existing network offerings with "hotspot" services based on wireless LAN technologies. These hotspots recognize that a large class of mobile users require broadband connectivity in locations that are semi-static workspaces. Roaming on a metropolitan scale, with high speeds and dynamic cell handoffs is not required. Instead, mobility is confined to a large building, conference facility or a relatively small campus. In this respect, WLAN proliferation can be treated as a different "radio access network" (RAN) for proposed 3G services. This integration of inherently insecure WLAN access networks with the billing, authentication, and management mechanisms of conventional telephony networks raises several issues which require in-depth discussion and evaluation.

Regardless of the many challenges, wireless LAN (WLAN) technologies are emerging as a compelling access method for next-generation communications. The interaction of these technologies with conventional PSTN and wireless telephony presents a clear disruption in the planned progression of wireless telecommunications. With appropriate support from network infrastructure, emerging multimode devices may even enable automatic, transparent roaming between physical networks [16]. Such "seamless roaming" may grow increasingly popular as WLAN hotspots appear in public locations such as airports, malls, and even your local telephone booth.

3.3 Wireless LAN Integration Issues

A simplified wireless telephony architecture is shown in Fig. 7. In this figure, an 802.11 "location area" is depicted as an alternative Radio Access Network (RAN) for the core provider network. This architecture is applicable to all 802-style WLAN access networks, including those with different modulation schemes and effective bit rates (802.11a/b/g/h) as well as those relying on port-based access control through higher-layer authentication (802.1x).

In comparing the carrier-integrated WLAN architecture with conventional telephony, the Access Point (AP) essentially performs many of the functions of the Base Station Transciever (BST) or Node B. Additionally, the Access Point Controller (APC) performs many of the functions of a Base Station or Radio Network Controller (BSC or RNC), and depending on packet forwarding architecture, may perform many of the functions of a GPRS Service Node (SGSN, GGSN) and/or a

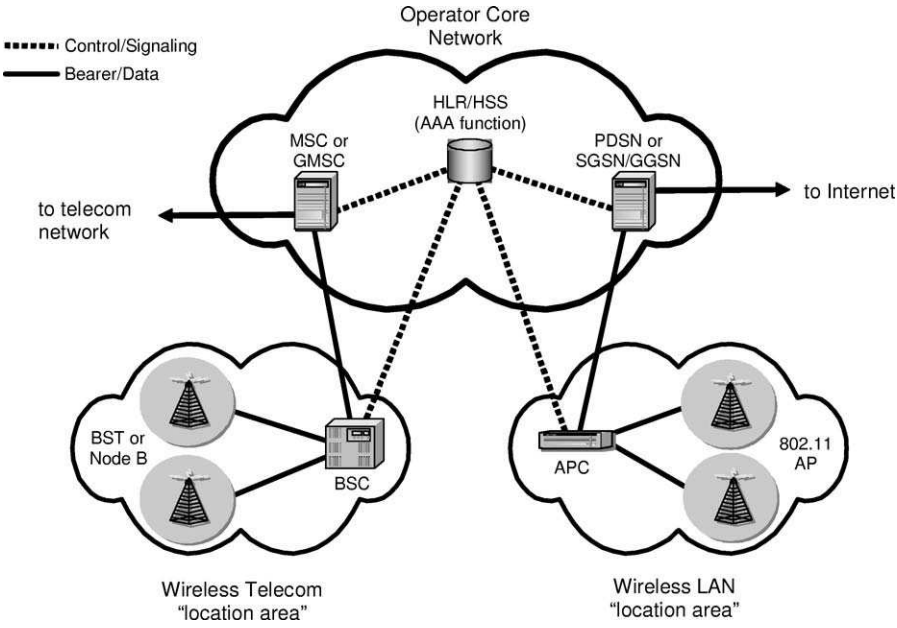


FIG. 7. Diagram of general wireless telephony network with 802.11-based RAN.

Mobile Switching Center (MSC). This integrated architecture infers some "carrier grade" requirements on all parts of the heretofore "commercial grade" WLAN infrastructure components.

Many challenges lay ahead in the integration of WLAN technology with conventional wireless telephony. These challenges can be loosely classified according to categories such as User Experience, Security and Access Control, Network Management, and Billing. Following sections discuss important aspects of each of these areas.

3.3.1 User Experience

In addition to differing wireless access technologies, wireless telephony and WLAN networks also have significant differences in user experience. For instance, the evolution from voice-only networks means that data/content are likely to originate inside the infrastructure of the network operator. For example, in the highly successful *i-mode* service in Japan, most delivered content is hosted by the network operator. The availability of content inside the infrastructure of an operator and the hierarchical organization of the network means that access to the public Internet may

be restricted to a few selected gateways. Nevertheless, at the expense of poorer service to data from the public Internet, more predictable service can be expected from data originating from within the operator's infrastructure. In contrast, independent 802.11 networks are often setup for the sole purpose of wireless Internet access, and an experience similar to that of a wired network, with plenty of content and service variation.

To this point, authentication methods for WLAN hotspots have been primarily browser-based, which presumes that the user's main interest in network access is for web surfing. In fact, industry consortia promoting WLAN usage have recommended this mode of access as the "best common practice" for most roaming WLAN users [17]. However, browser-based authentication, though convenient and user-friendly, is vulnerable to relatively simple theft-of-service attacks [18]. Additionally, as WLAN usage becomes more prevalent and adapts to different usage models, the user experience will benefit from network-based services such as automatic, transparent login and network selection, dynamic session handoff, and single point of contact for billing issues.

3.3.2 Network Security and Access Control

Wired LANs, which are typically implemented and managed by enterprise experts, tend to be physically secured by externally imposed policies and procedures (security guards, locked wiring closets, etc.). However, this level of physical security may not be available at all WLAN hotspots locations. Smaller hotspot locations where demarcation zones between "the core network" and "the access network" are not cost-justified may be particularly troublesome. Examples of such locations might include "Mom & Pop" coffee shops, small business DSL subscriptions, or even telephone booths. In these cases, the absence of physically verifiable access control for WLAN equipment means that critical WLAN functions must be remotely verifiable in a reliable, quick, and convenient fashion. Technologies which accomplish these functions in a cost-effective manner without undue infrastructure requirements have not yet been implemented in commercial-grade WLAN equipment.

3.3.3 Network Management

In keeping with the general defining characteristics of the IP network, most architectures for WLAN hotspots concentrate intelligence at the outer edge of the provider's network. In this case, the outer edge can be defined as the antenna itself (AP), the aggregation point for multiple access points (APC), or some combination of the above. Regardless of the specific architecture deployed or the configuration of equipment, the issues of manageability and security for a multiplicity of complex devices will be crucial for acceptance by network operators.

To conform to usual requirements, the setup, configuration, and other operational parameters of the devices must be externally manageable using conventional (industry standard) remote management technologies. Most telephony operators depend on highly centralized management facilities which seek to minimize "hands on" troubleshooting of deployed equipment. This tendency may be aggravated by the low cost and potentially low reliability of commercial-grade WLAN equipment.

Additionally, the indeterminate state of IP-based QoS and WLAN MAC-layer QoS technologies may complicate or defer large-scale rollouts. Complexity in the interworking of these disparate standards will certainly increase the difficulty of fielding relatively reliable, predictable, and autonomous "carrier-grade" solutions.

3.3.4 Billing

A primary hindrance to the integration of WLAN service may be the more pedestrian problem of *billing*. Customers may be impeded from using these services due to lack of a uniform billing arrangement akin to the familiar calling plans for cell phones. These cellular plans provide for a fixed usage model for a reasonable cost or a fixed charge per unit of usage (i.e., per minute charge) no matter where the customer is in the calling plan area (nation-wide or regional). It can be argued that the advent of these uniform predictable-cost plans have contributed to the explosion of cellular phone use.

Another example of "familiar" billing plans is the calling card model for wired telephony that enables a roaming user to use a telephony network while being billed at a fixed rate by a known carrier. This model rapidly replaced the direct-pay calling card model where the actual cost was determined by the policies of the local network to which the phone was connected.

The user of WLAN data services is no different. Users are reluctant to provide *carte blanche* to a wide variety of independent WLAN hot spots where the billing may be ambiguous or excessive. The observation follows that a successful WLAN service should follow a cellular phone billing model where the plan either has fixed units per month or a well defined, uniform usage cost. In fact, the authentication and billing infrastructure of the cellular phone provider could be directly used via a simple calling card model. The user logging onto the network is presented the option to choose a wireless carrier for billing and to submit a calling card number and PIN. This information is routed to the appropriate Line Information Database (LIDB) to provide authentication that is necessary to guarantee billing. With this approach, the network operator could leverage billing arrangements with local hot-spot operators similar to wireless telephony models that allow transparent roaming across multiple independent networks. A reasonable billing model for roaming WLAN data access might include a slight modification to the traditional provider approach to "off peak"

and "peak" minutes. In the case where authentication is tunneled or enabled through a WLAN hotspot, the provider may offer plans which include a large amount of "data minutes" in addition to the usual voice minutes. For example, a fixed-rate plan might enable 1000 peak voice minutes, 5000 off-peak voice minutes (nights, weekends), and 50,000 data minutes.

3.4 Important WLAN Technologies and Trends

Following sections discuss in some detail a few of the technologies and trends that will affect the integration of WLAN access networks with existing telephony networks. The capabilities (or limitations) of these technologies will impact the security, scalability, and QoS capabilities of integrated WLAN networks. In particular, the WLAN physical and MAC layers (OSI layer 1 and 2) are fraught with a variety of standards, technologies, and approaches which directly influence the viability of WLAN as an extension of the telephony network. Some key points of these standards are referenced heavily in following sections. So, the salient PHY and MAC aspects are summarized in Table III, and basic trends in the security and access control technologies are summarized in Fig. 8.

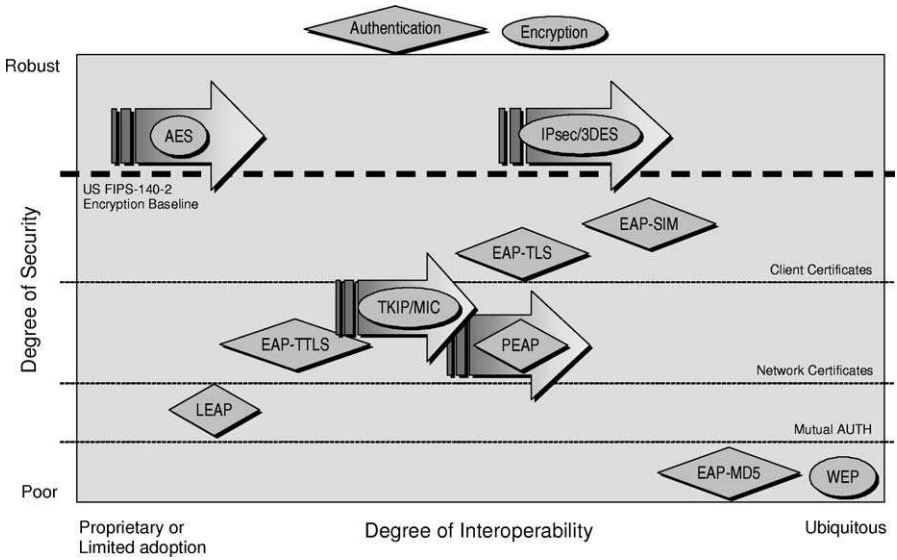


FIG. 8. 802.11 security/interoperability.

TABLE III
SUMMARY OF 802.11 PHY, MAC, AND ENCRYPTION STANDARDS

IEEE WG	Rate, RF Band, Chanels	Physical Layer Comment
a	6–54 Mbps 5 GHz UNII band 12 channels	• Uses Orthogonal Frequency Division Multiplexing (OFDM) in 5 GHz UNII band • Requires 2–4 times more APs for same coverage footprint as "b" • Incompatible with TPC/DFS spectrum management techniques of "h"
b	5.5, 11 Mbps 2.4 GHz ISM band 11–14 channels (3 usable)	• High-rate enhancement of 1 & 2 Mbps 802.11 Direct Sequence Spread Spectrum (DSSS) standard • Uses Complementary Code Keying (CCK) to achieve 5.5 & 11 Mbps • 2.4 GHz ISM band is unlicensed, crowded, & very sensitive to small changes in local environment • Only 3 channels are nonoverlapping (i.e., most are self-interfering)
g	22–54 Mbps 2.4 GHz ISM band 3 channels	• Higher "a" data rates in "b" (ISM) spectrum via use of OFDM • Backward compatibility mode with "b" (using CCK), but degrades throughput significantly • Two optional modulation modes for rates up to 24 Mbps: TI's PBCC & Intersil's CCK-OFDM
IEEE WG	**Description**	**Media Access Layer Comment**
e	MAC Layer Enhancements for Quality of Service	• Provided at layer 2—compatible with all 802.11 PHY (a,b,g) • Requires significant inter-working with higher-layer technologies to be effective • EDCF—Enhanced 802.11 Distributed Coordination Function (CSMA/CA with subslots & traffic classes) • HCF—Enhanced 802.11 Point Coordination Function (AP acts as bus-master, polling EDCF stations)
f	Inter Access Point Communication	• Inter-Access-Point Protocol (IAPP) enables roaming of 802.11 stations within IP subnets • On roam, new AP retrieves station/subscriber data from RADIUS server & session context from old AP
h	Spectrum Managed 802.11a	• Modifications for "a" to be acceptable in Europe (competes with HiperLAN2) • Frequency-agile "listen before talk" transmit mode (Dynamic Frequency Selection, DFS) • Uses minimum necessary power to transmit (Transmit Power Control, TPC)
i	Enhanced Security	• TKIP/MIC—Pre-standard WEP enhancement • AES—Standards-based WEP replacement

(*continued on next page*)

TABLE III — *Continued*

Technology	OSI Layer	Encryption Scheme Comment
WEP	OSI layer 2	• Asymmetric authentication only (mobile device to AP) allows rogue AP's • Serious implementation flaws (common, static key; no key management protocol; cleartext IV) render WEP almost useless
TKIP/MIC	OSI layer 2	• Temporal Key Integrity Protocol (TKIP) uses RC4 to pre-encrypt complete frames before WEP to obscure the IV. Shared 128-bit "temporal key" changes periodically; allows for per-user encryption keys (128-bit key + client's MAC + 16-octet IV) • Message Integrity Check (MIC) makes the CRC more robust to prevent undiscovered tampering
AES	OSI layer 2	• Advanced Encryption Standard (AES) replaces DES for "sensitive, unclassified" federal information (FIPS 197) • May require hardware refresh at client & AP for encryption co-processor
IPsec (VPN)	OSI layer 3	• Layer 3 implementations can be bandwidth & CPU intensive, and do not co-exist well with mobility protocols

Notes:

[a]The ISM band occupies 83.5 MHz (2.4–2.4835 GHz), and in the U.S. the UNII "lower" & "upper" bands occupy 300 MHz (5.51–5.35 GHz & 5.725–5.825 GHz).

[b]802.11c (Bridge Procedures) & 802.11d (Global Harmonization) are not particularly relevant here.

3.4.1 PHY Standards

The IEEE 802.11 standards for wireless LAN have evolved quickly to encompass a variety of technology areas as well as to address a plethora of problems. The WLAN physical layers definitions that are of interest here are 802.11a, b, and g. These standards specify two separate, unlicensed and unregulated frequency bands where WLAN equipment can operate. As indicated in Table III, 802.11 "b" and "g" operate at different effective data rates in the same 2.4 GHz ISM band (Industrial, Scientific, and Medical), whereas 802.11 "a" operates at up to 54 Mbps in the 5 GHz UNII band (Unlicensed National Information Infrastructure).

Many of the transmission difficulties encountered in WLAN usage are due to the fact that the popular 2.4 GHz ISM band is very crowded and sensitive to small changes in the local environment. As a result, RF interference and multipath distortion can be quite severe. Additionally, although 802.11b specifies a number of separate channels, the proximity of these channels combined with the relatively low quality of the electronics leads to significant interference, and only 3 of the channels are effectively usable. To combat RF interference and improve response to multipath distortion, the higher-rate "a" (5 GHz UNII band) and "g" (2.4 GHz ISM band) standards use Orthogonal Frequency Division Multiplexing (OFDM). OFDM is a

spread-spectrum technique where bearer data is distributed and transmitted in parallel via several precisely-spaced carriers.[1] Sometimes called Discrete Multi-Tone (DMT) modulation, this technique is also used in Digital Subscriber Loop (DSL) implementations.

Although "b" and "g" exist in the same RF band, they are not directly compatible since "b" uses Complementary Code Keying (CCK) rather than OFDM. The specification for "g" describes two required modes: OFDM for higher "a" rates in the 2.4 GHz ISM band, and CCK for backward-compatibility with "b." Additionally, "g" specifies two optional modulation modes to accommodate proprietary implementations at rates up to 24 Mbps: Intersil's combined CCK-OFDM, and TI's Packet Binary Convolutional Code (PBCC, also called "b+"). Unfortunately, legacy "b" devices will see "g" transmissions as noise, and although a pure "g" environment may be capable of TCP throughput up to 20 Mbps (4× effective "b" rates), all devices in a mixed environment may settle to the lowest common denominator. In the case of environments with "b" devices, this could lead to throughputs of less than 1 Mbps.

In the U.S., recent Federal Communications Commission proposals have recommended the allocation of more spectrum for "a" devices. In the current allocation, 300 MHz is available in the 5 GHz UNII band, with 11 channels split among "lower" & "upper" subbands. Recent FCC proposals would add 255 MHz in middle band (5.470 GHz to 5.725 GHz) and increase to 24 the number of available channels. The proposed allocation would also match the 5 GHz "middle band" allocated in Europe, resulting in some worldwide commonality in 5 GHz subbands. According to the proposal, transmitters in the new "middle band" would have to use advanced techniques such as TPC and DFS of "h" (see Table III) to avoid collision with RADAR transmissions. A side-effect of the additional channels could be the designation of special transmission regions for QoS-sensitive streams or out-of-band control information for bearer data in existing upper/lower UNII subbands.

3.4.2 MAC Standards

IEEE 802.11 Media Access Control (MAC) standards, provided at OSI layer 2, are compatible with all 802.11 physical layer (PHY) standards which are provided at

[1] OFDM overlaps each of the carriers in a channel rather than separating them with a guard band. The precise spacing of the carriers allows some orthogonality between the closely packed signals, thus minimizing intercarrier interference and improving throughput. For instance, 802.11a defines several channels, each 20 MHz wide. Each "channel" is an OFDM-modulated signal containing 52 overlapping, 312.5 kHz-wide carriers. Most of the carrier signals transport data, with 4 reserved for pilot signals. Raw data rates for each carrier can reach over 1 Mbps, if appropriate modulation and error-correction techniques are used. As a result, the composite (multicarrier) signal in one of the 20 MHz channels can have a data rate up to 54 Mbps.

OSI layer 1. Existing 802.11 devices achieve a loose coordination between distributed wireless nodes via an implementation of the MAC protocol known as Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA). Unfortunately, this loose coordination scheme doesn't differentiate between data streams or stations, so specific delay and bandwidth requirements are not supported. To partially address this issue, IEEE 802.11e specifies MAC-layer enhancements for QoS based on extensions of existing 802.11 mechanisms. The "e" recommendations may require significant inter-working with higher-layer technologies to be effective, particularly for carrier-integrated WLAN access networks.

3.4.2.1 Distributed Coordination—Contention-Based Access. The asynchronous, best-effort transfer provided by CSMA/CA is also known as the 802.11 Distributed Coordination Function (DCF), and is the basis of all WLAN media access schemes. With the CSMA/CA-based DCF, transmitting stations first test for an idle channel, then wait a random number of *slot times* before transmission to avoid collisions.[2] With this "listen before talk" scheme, successful transmissions must be positively acknowledged by the receiver. Without a positive acknowledgement, the transmitting station assumes that a collision has occurred so it enlarges its range for selecting random *slot times* ("contention window") before retransmitting the frame. With this randomized *backoff procedure*, the station whose backoff time happens to expire first during an idle period is allowed to transmit.

To provide differentiated access to the wireless channel, the Enhanced Distributed Coordination Function (EDCF) of 802.11e generalizes this approach to a hierarchy of "virtual" and "real" CSMA/CA stations with multiple subslots and traffic classes. With EDCF, a transmitting station employs a form of Class-Based queuing with a limited number of *Access Categories*. Frames from these *Access Categories* contend for virtual transmission opportunities using a CSMA/CA "listen before talk" mechanism as if they were independent DCF stations. However, EDCF *Access Classes* are assigned different contention windows and minimum interframe spacings so that, in most cases, backoff timers for frames from "higher priority" classes will expire sooner than other classes. As a result, "high priority" frames from EDCF stations will be able to contend for physical medium access with other stations more often than frames from "lower priority" classes.

3.4.2.2 Point Coordination—Contention-Free Access. Legacy 802.11 MAC standards specify a contention-free mode where privileged stations are polled for pending data frames. During this Point Coordination Function (PCF)

[2]WLAN *slot times* are a function of the physical layer. For instance, an 802.11b *slot time* is 20 μs, and an 802.11a *slot time* is 9 μs. The initial "channel sensing" time (distributed inter-frame spacing, or DIFS) is also a function of the PHY layer, and is 50 μs for 802.11b and 34 μs for 802.11a.

mode, which alternates with the purely contention-based DCF mode, stations can only transmit when they are polled by the access point. Unfortunately, the PCF implementation is optional and built on top of the DCF mechanism, and is unable to coordinate with the offered load of WLAN stations. As a result, PCF performance for QoS-sensitive applications is very poor [19]. The 802.11e specification recommends a "hybrid" PCF mode (HCF) which operates in both contention-based (EDCF) and contention-free modes to improve polling performance. HCF combines some DCF and PCF functions with QoS-specific polling and frame subtypes to achieve limited QoS transfers. With HCF, the WLAN access point has a shorter "listen before talk" wait-time than the other stations. This results in an effectively higher channel access priority that can be used to poll stations for QoS requirements or deliver QoS-sensitive frames. The HCF can also "reserve" follow-on transmission slots to concatenate contention-free bursts [20].

3.4.3 Access Control and 802.1x

Access control to a managed network is critical to maintaining network stability. IEEE 802.1x [21] is the "next generation" authentication architecture for WLAN networks. In the 802.1x architecture, the user's network access is limited to rudimentary functions until the authentication process is completed. With this architecture, user credentials can be centrally controlled and network access can be gated based on the result of authentication. There are many forms of authentication used in WLAN networks today. A few of these technologies are summarized in Fig. 8 and Table IV.

In the 802.1x framework, the Extensible Authentication Protocol (EAP, RFC 2284) is the dominant technology for coordinating the authentication conversation. EAP is a generalization of IP's common point-to-point protocol (PPP, RFC 1661) for basic authentication. As indicated in Fig. 8, protected EAP (PEAP), the combination of EAP with Transport Layer Security (TLS)[3] will likely become the de-facto authentication scheme for 802.1x. Tunneling legacy authentication methods, such as those based on username and password, via TLS security associations is important for preventing eavesdropping for user credentials as well as for allowing the client device to reliably authenticate the network. Network authentication via digital certificates is important in WLAN environments due to the extremely low barrier against man-in-the-middle attacks where illegitimate access points are used to capture credentials or hijack sessions. When EAP conversations are tunneled via TLS, the resulting authentication conversation has two phases. The first phase consists of establishing an encrypted connection (protective tunnel) where the network is authenticated to the client via digital certificates. The second phase of authentication

[3]TLS (RFC 2246, 3546) is the official IETF version of the Secure Socket Layer (SSL) protocol for strong encryption between client/server systems.

TABLE IV
SUMMARY OF POPULAR EAP TYPES

EAP Type	Comments
Message Digest MD5	• MD5 hash of username/password handed to RADIUS server • No facility for mutual authentication between device & network (rogue access points possible) • No management or dynamic generation facilities for WEP keys
Transport Layer Security (TLS)	• Client & Network Certificates for mutual authentication requires X.509 certificates on all clients (difficult to manage) • Some proprietary certificates have "Extended Key Usage" field which may be incompatible with EAP-TLS components from other vendors • Certificates can be used to generate dynamic per-user, per-session WEP keys
Tunneled TLS (TTLS)	• Mutual authentication using only network-side certificates • TLS tunnel protects PAP/CHAP-style username/password over EAP or other EAP methods for user-to-network authentication • Certificates can be used to generate dynamic per-user, per-session WEP keys
Lightweight EAP (LEAP) or EAP-Cisco	• Supports mutual authentication without dual certificates • Proprietary; requires end-to-end single-vendor and special client software • Supports dynamic WEP key generation with tunable duration
Protected EAP (PEAP)	• Mutual authentication using only network-side certificates • TLS tunnel protects "inner" EAP conversations for user-to-network authentication
Subscriber Identity Module (SIM)	• Mutual authentication via client & network "hardware certificates" (SIM) • Requires EAP interface to GSM SIM on client

uses the legacy authentication protocol "inside" the TLS-encrypted tunnel to authenticate the client to the network. As indicated in Table IV, Tunneled TLS (TTLS, [22]) and Protected EAP (PEAP, [23]) are the primary TLS-secured, two-phase authentication techniques proposed for use in WLAN environments. TTLS and PEAP are very similar, but TTLS allows a wider variety of "inner" authentication protocols such as CHAP (RFC 1994), PAP (RFC 1334), MS-CHAP (RFC 2433) and MS-CHAPv2 (RFC 2759), whereas with PEAP, the tunneled protocol must also be an EAP method. Additional EAP-types are summarized in Table IV.

In 802.1x, the participants in an authentication conversation are the *Supplicant*, *Authenticator*, and *Authentication Server*. Before the *Supplicant* is able to use upstream network services, it initiates an authentication request via EAPoL (EAP over LAN), a point-to-point link-layer protocol which encapsulates EAP packets into 802.11 frames. The *Authenticator* (usually the 802.11 AP) acts as a blocking gateway between the *Supplicant* and the network, only translating between EAPoL (EAP/802.11) and EAP/RADIUS to enable the authentication conversation with the

Authentication Server. Prior to successful authentication and authorization, the *Supplicant* is in an unauthorized state. In this state, the *Supplicant* node doesn't have a valid IP address or a gateway address to which to forward packets, and all IP traffic is denied by the *Authenticator*. In this fashion, the authentication process is accomplished without dynamic allocation of IP addresses at the WLAN hotspot, which alleviates potential security holes. With this architecture, the *Authenticator* is slaved to the result of the authentication process, which is handled completely by the upstream *Authentication Server*. Since RADIUS (RFC 2865) is the de-facto AAA technology for ISP's, the *Authentication Server* is typically a RADIUS server with subscriber information in a local database. Additionally, the interworking of EAP with RADIUS is well-defined (RFC 3579), and several authentication protocols can be mediated via the EAP/RADIUS combination, including public keys, digital certificates, and others as indicated in Fig. 8 and Table IV.

Integration with existing telephony authentication infrastructure may impose additional requirements on WLAN users. For example, wireless telephony networks such as GSM/GPRS rely on an authentication procedure involving a "Subscriber Identity Module" (SIM) installed on the mobile handset. An example of signal flow and protocol stacks for SIM-based authentication in a WLAN access network is shown in Fig. 9. Note the adaptation of an interface between the WLAN zone, which is strictly IP-based with 802.1x and EAP, and the operator's core network, which may use a combination of IP protocols and SS7-based systems for authentication, subscriber information, and billing purposes.

In addition to simple authorization (i.e., network access), 802.1x can be used to provide privileges to users, nodes, and even particular network services. Gated by the authentication process, 802.1x enables capable network elements to modify access privileges according to individual entitlement. Although RADIUS-authenticated sessions can already provide limited service authorization, 802.1x creates a robust framework for service configuration, resulting in the potential for fine-grained, dynamic privilege authorization.

3.4.4 Over-the-Air Security

Wired-Equivalent Privacy (WEP) is the over-the-air encryption scheme built into current 802.11-style WLAN networks. The WEP architecture was specified incorrectly, and this has led to a rash of security problems in WLAN deployments. The general WEP architecture is shown in Fig. 10. Note from the figure that the primary issue with WEP is the use of an unencrypted 24-bit initialization vector (IV) in the header of each encrypted frame. The per-packet IV is used by the receiver, in conjunction with the master WEP key (shared secret), to reconstruct the specific key sequence which was used to encrypt the payload of the frame. The cleartext transmission of the IV incrementally reveals information about the WEP shared secret. Once

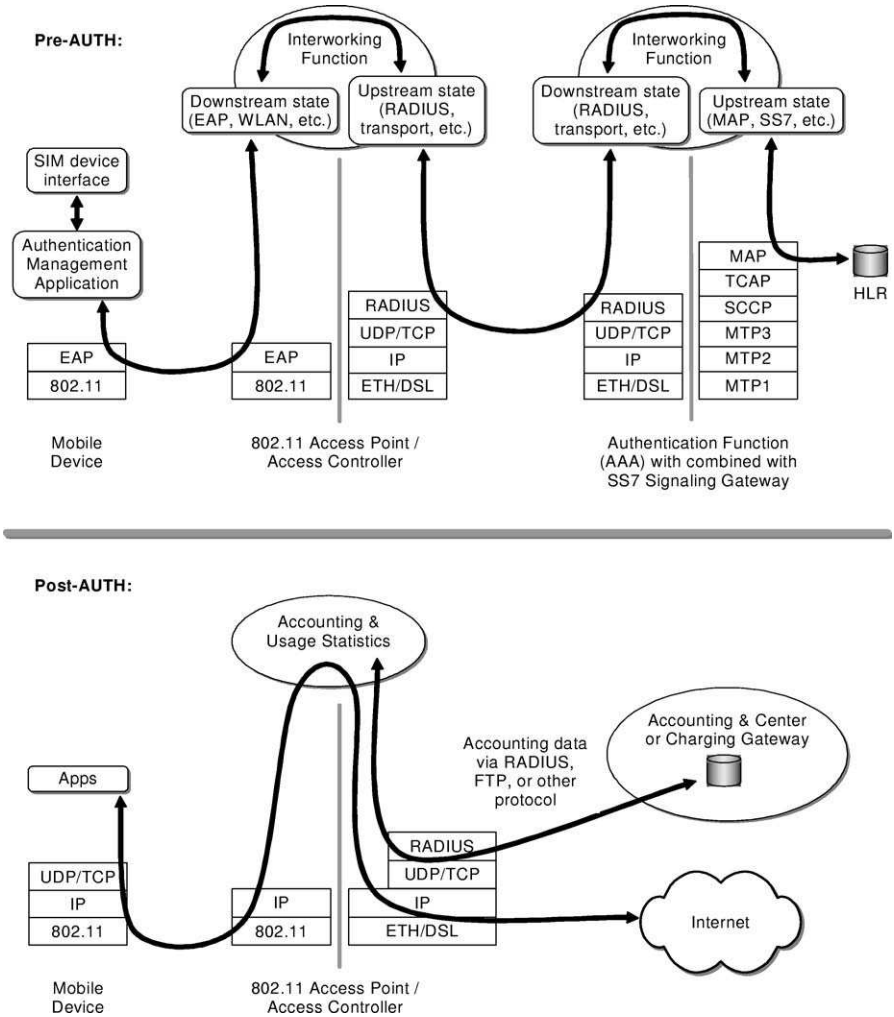


FIG. 9. Signaling, protocol, and application diagram for 802.1x WLAN authentication process. The diagram shows signaling flow for the pre-authenticated and post-authenticated states.

the WEP shared secret is known, each subsequent frame can be immediately decrypted [24]. The implementation flaw appears when the same WEP shared secret is used to create encryption keys which encrypt different payload information using the same IV. This occurrence of "IV reuse" creates a single instance of a compromisable

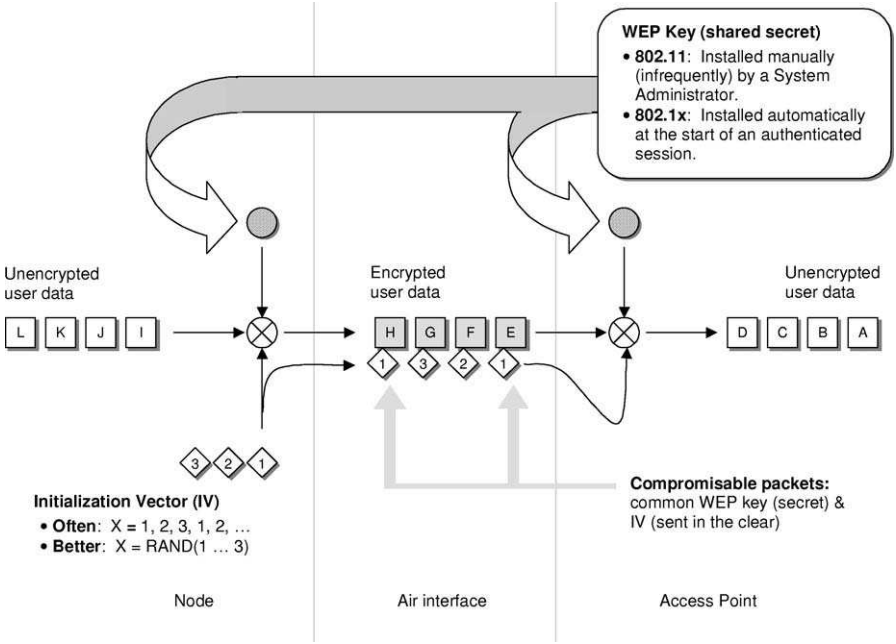


FIG. 10. Diagram of WEP usage in 802.11 and 802.1x. Note that cleartext transmission of the 24-bit IV for each packet, whether chosen randomly or deterministically, can be used to reveal information about the WEP secret key.

frame. The number of unique IV's is limited, and leads to a high likelihood that many compromisable encrypted frames will be produced in a short time period. As a result, WEP-secured WLANs are vulnerable to attacks which collect frames and cross-reference their known fields (IP numbers, etc.) based on reused IV's. WEP-cracking techniques will become even easier as WLAN air speeds increase to 54 Mbps and beyond simply because more encrypted frames, and thus more compromisable frames, will be available in a shorter period of time.

To address the WEP problem, IEEE 802.11i has recommended more sophisticated over-the-air encryption methods such as Temporal Key Integrity Protocol (TKIP) and the Advanced Encryption Standard (AES). Although the technologies summarized in Table III and discussed in following sections will improve the quality of over-the-air encryption for WLANs, it is likely that conventional WEP will coexist with these newer approaches for some time. This is primarily due to the fact that a complete hardware refresh will be required for full implementation of 802.11i with AES. This is complicated by the fact that complete ratification of 802.11i may not occur until a

significant installed base of both WEP-only and TKIP-capable systems has already been fielded.

3.4.4.1 Temporal Key Integrity Protocol (TKIP). TKIP, the precursor to complete 802.11i, is an improvement over conventional "static-key" WEP. In "static key" WEP, a common master key is distributed and changed infrequently, and is subject to the class of exploits described by Fig. 10. To combat these issues created by use of common, pre-configured WEP shared secrets for all devices, TKIP assigns a unique WEP key for each user dynamically at the start or re-authentication of a session. With this approach, the WEP keys in-use at any time should be unique for each user on a particular WLAN access point. However, with faster over-the-air data rates the likelihood of IV reuse can again become a problem. This may lead to approximately the same collection of compromisable encrypted packets. In addition to rotating WEP keys, the strengthened Message Integrity Check (MIC) procedure makes the encrypted frame more robust against undetected tampering.

3.4.4.2 Advanced Encryption Standard (AES). AES is an approach to encryption of "sensitive, unclassified" information that has been adopted as a U.S. Federal Information Processing Standard (FIPS 197) [25]. The AES algorithm is a symmetric block cipher for encrypting and decrypting 128-bit data blocks using cipher keys with lengths of 128 (AES-128), 192 (AES-192), and 256 (AES-256) bits. AES is based on the *Rijndael Block Cipher* algorithm, where a state matrix of bytes, or intermediate cipher result, is iteratively processed in "rounds" via a succession of nonlinear byte substitutions, cyclical row shifts, and linear column transformations. The number of "rounds" is a function of the input block length and the size of the cipher key. *Rijndael* is particularly useful for encryption of layer-2 WLAN transmissions because it is parallel by design, and can lend itself to highly efficient implementations (minimal code, minimal memory, fast performance) suitable for embedded devices. Additionally, the algorithm is "self supporting" in that it does not borrow sub-algorithms from other cryptographic schemes or procedures, and it makes no use of arithmetic operations or well-known non-repeating sequences.

3.4.5 Wireless QoS

Wireless multimedia streaming is becoming a reality with high-speed wireless data connectivity. In fact, multimedia streaming is a key goal of 3G, and handset specifications for handling streaming media have been developed [26]. Nevertheless, challenges still remain in streaming media over intrinsically unreliable wireless links. Generally, streaming media applications are sensitive to packet losses, delay, delay-jitter, and throughput—areas that are not well addressed by wireless networks.

For less demanding streaming applications that can tolerate a few seconds of setup delay, the challenge of streaming media is reduced to maintaining "good-put." However, even in the latter case, additional mechanisms may still be necessary to ensure positive streaming media experience.

Besides classical techniques involving forward error correction, ARQ, and diversity schemes in the physical and link layers, there are recent works that may enable better wireless streaming without violating the lower layers. For instance, in an 802.11b network, getting half the data from two access points can sometimes achieve better performance than getting all the data from the closer access point due to better insurance against fast variations in link quality [15]. For 3G networks where the wireless link has long latency, an important challenge has been providing fast feedback to the transmitting end to effect adaptation. To this end, work in network agents at the interface between wired and wireless network can provide meaningful improvements to streaming quality [27,28]. The purpose of such network agents is to provide a fast but approximate feedback to the transmitting end to effect adaptation.

Central to media streaming is the requirement of sustained throughput. When fluctuation in available bandwidth is frequent and drastic, e.g., switching to lower modulation format in response to poor wireless link quality, mechanisms to accordingly adjust media rate is required. There has been extensive research in methods to produce rate-scalable representation of video. Nevertheless, commonly deployed media streaming systems typically employ the more mature non-scalable representation of video. For instance, both 3GPP and ISMA currently prescribe the use of MPEG-4 for video. The ability to cater to different channel conditions is relegated to switching between multiple copies of the same content compressed at different bit-rates. In the future, dynamic adjustment of MPEG encoding properties such as picture size and bit-rate, coupled with fast feedback, may be a key component to guaranteeing throughput on a wireless network.

4. Commodity Software

The growth of the Open-Source community development model has "disrupted" the typical process of development and deployment for many industries, including telecommunications. This dynamic development model, driven by Internet-based and Open-Source collaboration tools, allows software projects to be initiated rapidly and completed efficiently via worldwide teams of developers. In addition to the clear improvements in process efficiency, this new development paradigm has leveraged the pipeline of global timezones, transparency of source code, availability of documentation and support, and real-time technical discussions to produce superior community-owned products. Examples include world-class software and systems

such as the Apache web server [29] and the Mozilla web browser [30], as well as the Linux kernel itself [31].

In the telecommunications sector, the general popularity of Linux has led to a groundswell of support for "Carrier Grade" extensions to the Linux kernel and operating environment. Following sections discuss some of the most important facets of Linux and its application in telecommunications. In addition, issues of highly-available systems (which are critical in telecom applications) are discussed in the context of Open-Source technologies and initiatives.

4.1 Linux: Open-Source Poster Child

The widespread momentum enjoyed by Open Source technologies is succinctly expressed in the community development of the Linux operating system. The clear evidence of support for Linux by most computer industry leaders and at all levels of the system stack (application software, middleware, kernel modifications, hardware and systems, etc.) indicates a "flashpoint" of acceptance in the telecommunications industry. This phenomenon is increasingly true even though there are some subtle differences between "open source," as conventionally interpreted and defined, and "free software," as defined by the Free Software Foundation (FSF) [32]. In the FSF definition, access to the source code ("open source") is a precondition to the four necessary freedoms[4] that define "free software." Issues are often encountered, however, with the *Copyleft* position of the FSF's "GNU General Public License" (GPL). In a form of irrevocable inheritance, *Copyleft* requires that software released under the GPL (and variants) must be similarly licensed. As a result, all modified and/or extended versions of "free software" must also be "free software," which implies "open source." This condition is often viewed as being contrary to commercial aims, where proprietary mechanisms (algorithms, software implementations, etc.) must be protected or "closed."

Interestingly, the Linux kernel seems to have a unique position with respect to these very different perspectives on software licensing. Although most Linux applications and kernel components are fully licensed under the GPL, the Linux kernel itself is released under a slightly modified form of GPL which allows any software to freely use kernel interfaces without being subject to GPL inheritance. So, one of the most interesting aspects of Linux, which is enabled partially by its "special" licensing considerations under the GPL, is that the kernel is highly modular and designed for arbitrary extension.

[4]The FSF "freedoms" are freedom of execution (use), freedom of examination (study), freedom of endowment (redistribution), and freedom of enhancement (modification).

4.1.1 Linux Kernel Modules

The widespread popularity of Linux as an alternative operating system for enterprise as well as technical computing is well-documented in the trade publications. One of the fundamental aspects of Linux that makes it amenable to interesting system architectures and flexible configurations is the use of dynamically loadable kernel modules. Linux kernel modules (LKM's) can interface freely with the Linux kernel due to the loophole explicitly stated in the modified GPL statement issued with the kernel distribution [31]. As a result, "aftermarket" modules can redefine or extend particular kernel functions in a proprietary fashion.[5]

This modular architecture is particularly appealing for device drivers and other capabilities that may need to be selectively loaded during system operation. However, because of the close association with the running Linux kernel, an LKM can be made to do some interesting and potentially dangerous things. For example, the system `write()` call can be redefined during module initialization so that all invocations of `write()` are intercepted, parameters of the invoking process are validated against a simple policy rule, and the buffer of data is modified before being transferred to the "normal" `write()` function [34]. Although LKM's can easily snoop the current task, filter or override specific system calls, and access user-space memory, they can also be used to harden kernel behavior by detecting failure modes and beginning failover procedures, or to trigger special processing on data entering/leaving the system.

An example of a simple LKM is shown in Fig. 11. When an LKM is loaded into the running Linux kernel, the kernel invokes the module's "constructor" function, in this case via `module_init( setHook )` which initializes the module and prepares it for execution (allocates memory, etc.). Likewise, upon unloading of the module, the "destructor" function is invoked via `module_exit( removeHook )` so the module can clean up after itself (deallocate memory, reassign functions, etc.). Between the execution of the "constructor" and "destructor" functions, the body of the module, including kernel symbols, functions, data, etc. are bound into the running kernel and accessible by user-space applications and kernel operations. In the case of Fig. 11, the module registers itself as a callback function in the kernel's NetFilter packet-processing subsystem, which is explained in detail below.

4.1.2 Packet Filtering—NetFilter

The NetFilter framework is an example of the flexibility of the modular Linux kernel. NetFilter allows packets traversing the Linux IP stack to be intercepted, snooped,

[5] Of course, nothing in Open Source software is absolute, particularly when it approaches mixing "closed source" with GPL-licensed software [33].

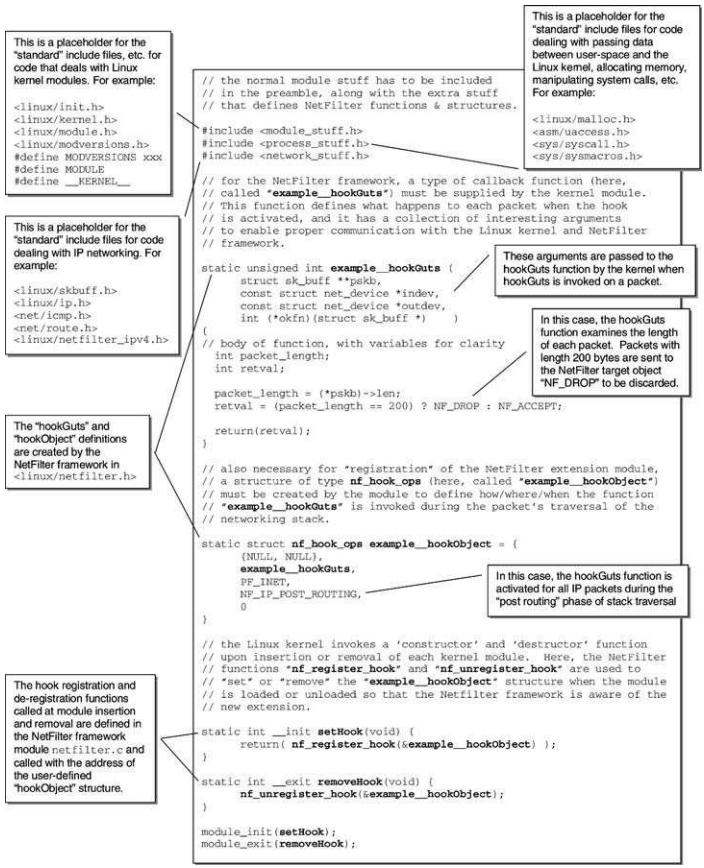


FIG. 11. Example of low-level extension module for Linux NetFilter (derived from various sources at http://www.netfilter.org).

or otherwise "mangled." The NetFilter framework is included in the standard Linux kernel distribution (2.4.x) with several user-space tools for creating arbitrary packet filters. If further manipulation is necessary, packets can even be diverted to user-space handlers and then re-inserted into the flow of packet-processing. The NetFilter framework is activated at five important points in the trajectory of IP packets, including the "pre-routing," "post-routing," "forwarding," "input," and "output" stages. When packets cross each of the five NetFilter-waypoints, registered callback functions can be invoked against the contents of admissible packets. Several extensions of the NetFilter framework have already been developed, including the `iptables` collection of utilities.

The approach to creating an LKM with the NetFilter framework is shown in Fig. 11. The figure contains an explicit (if useless in practice) illustration of a kernel module which uses the NetFilter framework to hook into the Linux IP stack and discard packets that match certain criteria. As shown in the figure, the "body" of the module contains two basic items: a `hookGuts()` function which is invoked in callback fashion against packets matching some simple criteria, and a `hookObject` structure which creates the context for `hookGuts()` to be registered as a NetFilter callback function. When the module is loaded, the "constructor" registers the callback function by passing the `hookObject` structure to the kernel via the NetFilter interface function `nf_register_hook()`. In addition to a function pointer to the `hookGuts()` callback, the `hookObject` contains initial packet matching and hook registration information. This information includes the protocol family to restrict which types of packets will be fed to this hook, the `hooknum` or traversal point in the IP stack that this hook is to be registered for invocation, and the relative priority of this hook with respect to other hooks registered at the same traversal point. In this case, the module enables a `hookObject` to operate on IP-type packets (`PF_INET`) during the "post-routing" (final) phase of IP stack traversal (`NF_IP_POST_ROUTING`). For packets in this phase, the `hookGuts()` function examines the length of each packet and discards or accepts the packet based on a simple test. The arguments passed to `hookGuts()` by the kernel include the hook number or traversal point at which it was invoked, a pointer to the socket buffer structure which contains the actual packet data and headers, etc., pointers to the device structures which identify the origination/destination interfaces for the packet, and a pointer to a user-space function for processing the packet contents. When the module is removed from the kernel, the "destructor" operation unregisters the callback function and `hookObject` structure via the NetFilter interface function `nf_unregister_hook( hookObject )`.

One of the main points of interest in the NetFilter framework (not explicitly shown in Fig. 11) is in the definition of the socket buffer structure `sk_buff` (defined in `<linux/skbuff.h>`). The collection of `sk_buff` structures is a standard linked list with information about each packet such as when the packet arrived, which interface and socket it belongs to, the packet headers (transport/network/link-layer), and some other control information. However, for Linux kernels compiled with support for the NetFilter framework (with kernel configuration variable `CONFIG_NETFILTER`), the `sk_buff` structures are "augmented" with a pointer to other `hookObjects`. This additional parameter creates the possibility for direct communication between different `hookObjects`, which allows for context-based manipulation of packets. As a result, NetFilter provides the modular Linux kernel with the sophisticated packet-processing capabilities necessary for next-generation telecommunications networks.

4.1.3 Preemption, Latency and Process Scheduling

Among the developments which have influenced the compatibility of Linux with the rigorous telecommunications environment, recent improvements in preemptibility and latency are among the most significant. These enhancements were introduced into the 2.5-series (development) kernels, and will become part of the mainstream Linux environment starting with the 2.6-series kernels.

Latency in a computer system can be defined by a lack of responsiveness to asynchronous events (interrupts). This is usually a function of several factors, including interrupt assertion/handling and process scheduling. In the Linux kernel, latency appears to be dominated by the elapsed time between interrupt service completion and process scheduler invocation. So, increasing the number of process scheduling opportunities during kernel execution ("more scheduling opportunities per unit time") is a viable approach to reducing latency. Among Linux kernel developers, two competing methods were proposed for achieving the goal of "more scheduling opportunities per unit time:"

- The "preemption patch," which sprinkles opportunities for the scheduler around sections of commonly executed kernel code, and
- The "low-latency patch," which inserts targeted scheduler opportunities in sections of kernel code that exhibit high latency.

The two approaches are almost completely complementary because the preemption patch uses *a priori* assumptions on kernel execution to increase the likelihood of scheduling, whereas the low-latency patch uses simulated loads in an iterative, *a-posteriori* fashion to custom-tailor the performance of a specific kernel. As a result, the low-latency patch requires much more effort to maintain. In either case, the notion of absolute guarantees on actual performance is subject to mismatch between assumed conditions and actual load. For the preemption patch, the critical assumed conditions are based on general principles of kernel execution, or probability of encountering certain conditions. For the low-latency patch, the critical assumed conditions are based on a likely system environment and simulated load, which drives a statistical analysis. In particularly sensitive cases where simulated system load modeling is reasonably accurate, *a-posteriori* techniques such as the low-latency patch can be used to isolate particularly obvious latency problems. To achieve very small latency values, confidence intervals/levels, or absolute maximums, it is very likely that a combination of these approaches would be required.

4.1.3.1 Preemption Patch. The preemption patch approach was introduced by MontaVista Software, and has been supported by kernel developer Robert Love [35]. With this approach, the Linux kernel is allowed to switch to another execution thread if it isn't executing a "critical" section of code. In other words, the

kernel can be preempted unless preemption has been specifically *forbidden*. This is in contrast to the normal Linux kernel which can only be preempted if preemption has been *allowed* by certain actions (i.e., a targeted scheduling request, as in the case of the "low-latency patch").

The preemption patches, which are part of the 2.5-series kernels, modify the Linux task structure, scheduler, spinlock primitives, and architecture-specific interrupt return routines to include a preemption flag. As a result, each time a spinlock is released or an interrupt routine returns, the kernel has an opportunity to invoke the scheduler. In the non-preemptible kernel, these opportunities didn't exist. So, the effect of the preemption patch modifications is reduced time between a scheduling request and the servicing of the request by the scheduler because "more scheduling opportunities per unit time" are available. For an empirical performance example, Williams [36] achieved better than 1 millisecond response time in 99.99% of scheduling requests, with a maximum latency of 45.2 milliseconds, and Love [37] achieved multiprocessor performance gains on the order of 700%.

4.1.3.2 Low-Latency Patch. The low-latency patches are the result of an after-the-fact analysis to find critical latencies in a specific kernel and defeat them by adding targeted preemption points. This is an attempt to approximate the goal of "more scheduling opportunities per unit time" by carefully picking the low-hanging fruit. For an empirical example of the effectiveness of these patches on the 2.4.17 kernel, Williams [36] achieved better than 1 millisecond response time in 99.9999% of scheduling requests.

However, there are a collection of issues with this sort of approach to system optimization. In addition to errors of omission that may occur since target latencies are identified through simulation, the effort to essentially custom-tailor every kernel for a deployment environment can be prohibitive, especially for telecom applications. Even in the case where this "bug-tracking" effort may be justified, there are difficulties associated with inserting test and recording mechanisms in the kernel to collect valid data without perturbing actual running conditions. It is also difficult to ensure that the test environment is physically & logically identical to the target environment. In cases where these factors can be isolated and compensated for, the optimized system performance can be quite good (if somewhat inflexible). Under the right conditions, the "low-latency patch" can produce much tighter bounds on maximum latency than the "preemption patch." For example, Williams recorded maximum latencies of 1.3 milliseconds for the "low-latency patch" versus 45.2 milliseconds for the "preemption patch" during initial testing. However, further testing of the "low latency patch" revealed spurious outliers of greater than 200 milliseconds [36]. This is evidence of the risk associated with a simulation-based optimization which relies on a lack of random events.

4.1.4 Linux Kernel Version 2.6

In addition to the improvements made in preemptibility and latency, starting with version 2.6 the Linux kernel contains several other improvements which will make this Open-Source alternative highly competitive in the telecommunications marketplace. These improvements affect the performance of the task scheduler, the block I/O layer of the kernel, the virtual memory management subsystem, and the ability of the kernel to handle process threads.

4.1.4.1 Scheduler. The improvements in process scheduling coupled with the enhanced preemption characteristics of the Linux kernel should result in deterministic environment for demanding applications. In a fashion akin to "per hop behaviors" in packet-switched networking, the process scheduler in an operating system determines which process will be allowed to use CPU time for the upcoming time-slice. The simplest approach selects the "best" process from among all available processes via an exhaustive search. In newer Linux kernels, this nondeterministic or "O(n)" algorithm has been replaced with a deterministic or "O(1)" approach which places fairly strict bounds on the execution latency of the scheduler. In other words, instead of performing an exhaustive search over all candidate processes, the scheduler simply selects the "best" process based on some predetermined ranking criteria such as priority-level and availability. This is similar to having the runnable processes pre-classified into queues weighted by priority level, and selecting processes from the highest priority queue first. In multi-processor environments, each processor maintains an individual list of prioritized tasks. This approach distributes priority contention over the collection of CPUs and allows for CPU-affinity so that tasks don't "bounce" between CPUs. As a result, the system exhibits more graceful multiprocessor scaling.

Although the O(1) scheduler is generally an improvement in the Linux kernel, it is not without some implementation issues related to "starvation." For example, two runnable tasks on the same processor with the same dynamic priority *should* exhibit approximately the same CPU utilization. However, with the Linux O(1) scheduler it is possible, under proper conditions, for one task to "starve" the other task by consuming almost 100% of the CPU time. Additionally, in multiprocessor systems, tasks maintain affinity to the last-used CPU and can't be rescheduled to run on another CPU until after this affinity period expires (usually a few milliseconds). As a result, the too-frequent use of mechanisms such as `sched_yield()` to manually achieve application-level fairness or thread synchronization can cause all tasks to have affinity for the same CPU, and lead to "starvation" between CPUs [38].

4.1.4.2 Block I/O. Improvements in block I/O to extended memory have also increased performance of Linux systems with appropriate peripherals. Some

peripheral devices, such as disk drives, transfer information to the operating system kernel in "chunks" (blocks) that depend on the physical characteristics of the device. Early versions of the Linux kernel staged these block transfers into an intermediate buffer before using a memory-to-memory copy to move the block data to its final target buffer. The block I/O enhancements to the 2.6-series kernels will improve throughput with block devices by eliminating the need for the intermediate copy stage. Additionally, improvements to the I/O scheduling process include a weighted-queueing approach to read/write tasks as well as distributed lock mechanisms for multiple devices. These enhancements will allow for more efficient multiplexing of I/O requests as well as more deterministic access to block devices.

4.1.4.3 Threading. Efficient handling of process threads is another significant area of improvement for the 2.6-series Linux kernel. Threads (sometimes called lightweight processes) are independent, sequential streams of execution within a process. Threads generally share the instruction and data space and file access mechanisms of the process, but have separate "execution state" described by a stack and register contents. Because of the shared context, swapping execution state between threads in a process is simpler than swapping between processes.

As part of a portable programming model, user-space threads are mapped to kernel threads by a threading library. In newer Linux kernels, Linux has user-space threads mapped in a one-to-one fashion onto kernel-space threads by the Native POSIX Thread Library (NPTL) [39]. NPTL makes threading performance on Linux more efficient by using existing processor-specific registers for saving thread context (where available) and leveraging concurrent kernel enhancements relating to synchronization (fast user-space locks, or "futexes"). This drastically reduces the overhead required in switching state context when changing threads. Improved conformance to current POSIX standards for synchronization as well as signal handling and grouping of threads has also contributed to improved thread-handling performance in Linux.

4.2 Carrier Grade Linux (CGL)

In addition to Linux kernel enhancements that are being made constantly by the open-source community, the Open Source Development Lab (OSDL) is hosting the specification and development of a specialized set of features for "carrier grade" systems [40]. The focus of the OSDL Carrier Grade Linux working group (CGL) specifications can be divided into three areas which are critical for telephony systems: Performance, Standards Compliance, and Reliability, Availability, and Serviceability (RAS).

Performance optimization is an important area for telecommunications systems. From processing real-time data streams to facilities for handling massive numbers

of events, processes, and data objects, telephony systems are exposed to some of the most stringent and varied performance requirements of all computing systems. As a result, many of the performance requirements drafted for CGL seek to optimize aspects of system behavior related to response time, tunable memory and process controls, and scalability. CGL specifications in this area are related to preemptibility and latency for near-real-time systems, improvements to the virtual memory system to allow selective control of page faults, CPU/process affinity in multiprocessor systems, and flexible scheduler policies to enable "tuning" system performance for processes with known priority characteristics. Fortunately, many of these important features, such as preemption, latency, and process scheduling are important in many application areas, and are already being addressed by the larger Linux kernel community (see Section 4.1.3).

Reliability, availability, and serviceability (RAS) are arguably the premier characteristics for telecommunications systems, which often must operate unattended for long periods (years) with no provision for failure or upgrade. These overall system characteristics are dependent on qualities derived from individual components as well as the interaction between integrated subsystems. As a result, the CGL requirements for RAS focus on consistent platform support for hot-swap subsystems (disks, IO, etc.), definition of remote boot/reboot and diagnostic capabilities (network console, kernel crash dump, analysis tools, etc.), and techniques for stable unattended system operation (application hearbeating, watchdog timers, centralized event logging, etc.).

Compliance with standards and a common approach to system construction is also important for computer systems, particularly in the area of interchangeable hardware subsystems and the portability of software. As a "lightning rod" for open-source development, Linux is naturally fairly compliant with most current computing standards (in particular, IETF standards). The CGL emphasis on additional standards focuses on fine-grained requirements for coding habits (avoiding unnecessary `panic()` calls), POSIX compliant interfaces for timers, signals, messaging, threads, and event logging, and important new IETF protocols such as IP version 6 (IPv6) and the Stream Control Transmission Protocol (SCTP). Additionally, a key goal of CGL is to promote interoperability between multiple software products, so compliance with the Linux Standards Base (LSB) [41] and Service Availability Forum (SAF) [42] specifications is critical for system and middleware interoperability.

4.3 Highly Available Systems

In addition to the basic platform and operating system facilities defined by CGL participants, telecommunications systems often use "middleware" to achieve extreme high-availability and failover mechanisms between redundant facilities. In

terms of functional elements in a telecommunications network, highly-available systems are often based on clusters of computers which perform a specific task.

In general, computer systems are clustered in order to guarantee access to certain critical services or system components. For example, the Internet is, in a sense, a very loosely coupled "cluster" of computers designed to distribute the flow of information and eliminate catastrophic system failure. More commonly, however, clusters are localized groups of computers designed to optimize a particularly important facet of a computing task. In this sense, the definition of "important task" might include the availability of data in a filesystem or database, the state and activity of a critical process or program, or the distribution of load associated with processing or dataflow. Clustering technologies can be described using three different perspectives:

- *Data availability*—Some clustering technologies are designed specifically to enhance the availability of data storage and retrieval subsystems. These technologies provide access to mass storage facilities via parallel paths for multiple computing nodes. Often, the computing nodes in the cluster are connected simultaneously to multiple storage devices via a switch fabric or other mesh interconnect, and the data on the storage devices is replicated to ensure survivability. Examples of Data Availability clusters include robust filesystems, storage clusters, storage area networks, and certain types of database middleware.
- *Efficiency*—Some clustering technologies are designed specifically to increase processing or data throughput, or to dynamically adjust the utilization of constituent subsystems. These technologies tend to have a strict hierarchy between computing nodes, and provide feedback mechanisms between master and slave subsystems to achieve load-balancing for processing, data distribution, or parallel operation on a single task. Candidate datastreams or processing tasks that are well-matched to this type of cluster tend to exhibit a high degree of parallelism, and can be cleanly decomposed into multiple independent sub-tasks or data elements. Examples of tasks amenable to Efficiency clusters include image processing operations, data mining, and other high-performance computing tasks. Server farms and other IP-based load-balancing schemes can also be classified in this way.
- *Process or task continuity*—Some clustering technologies are designed specifically to maintain the continuity of a process or computing task. These technologies provide oversight for high-priority tasks as well as the facilities for reinstantiating these tasks on other nodes (failover) in the event of a software or hardware failure. To accomplish seamless failover without loss of process state or internal data, application processes typically participate in a checkpointing protocol to mirror critical parts of internal data to a backup process in a separate computing node. The backup process may be idle (active-standby), processing

data only after a failure signal is given, or the backup process may be duplicating the effort of the primary task (active-active) and producing some output which is selected based on failure notification. Based on the status of the backup process, the checkpointing protocol can be executed periodically or on-demand (i.e., just before process termination), and can result in data transferal to an external device via a cluster-private network, or to a tightly-coupled ancillary processor via shared memory.

Clearly, a particular instantiation of a cluster exhibits various levels of Continuity, Data Availability, and Efficiency depending on the task for which it is optimized. In telecommunications applications, the importance of each of these factors is derived largely from the location in the network where the cluster is deployed, the nature and scale of the network, and the function which the cluster is intended to perform. For example, a cluster of computers might be involved in performing a call-processing application in a large call center. In this case, the availability of a process to handle incoming calls would be critically important to prevent "call blocking" or the appearance of system overload. Additionally, maintaining the processing state and data associated with each individual accepted call would be critically important to maintain transaction integrity and avoid re-initialization. As a result, a clustering solution with strengths in Efficiency and Process Continuity could be more appropriate than a solution with strengths in Data Availability. Regardless of the particular location or function, the role of clustering in telecommunications is both critical and application-specific. As a result, the predominant means of accomplishing highly-available systems for telecommunications networks has been via proprietary technologies. Today, the presence of robust Open Source solutions is "disrupting" this mainstay of telecommunications equipment vendors. In addition to the Linux phenomenon and OSDL Carrier Grade Linux projects, a sampling of other influential projects or initiatives in this space are outlined in following sections.

4.3.1 Telecom InterProcess Communication (TIPC)

The Telecom Inter Process Communication (TIPC) is a protocol specification and representative implementation (for Linux) that has been designed for highly-reliable intra cluster communication [43]. TIPC began as a proprietary high-availability technology, but has recently been ported to Linux as a loadable kernel module. TIPC was presented to the CGL community as a toolbox for development of carrier-grade clusters on Linux. Spurred by lobbying and community participation efforts, the TIPC baseline (concept, specification, implementation) seems to have been adopted as the primary candidate for CGL clustering.

Leveraging an SS7-like network concept, TIPC provides basic infrastructure for intra-cluster communications and process management. In TIPC clusters, processors

are grouped via a hierarchy of full-mesh interconnections which allows for a variety of architectural simplifications. For example, cluster interconnect networks tend to have high bandwidth with low message loss (short transfer time, few retransmissions), most messages are relatively short (less than 500 bytes), most communication is direct rather than routed through multiple hops (complete destination addressing is unnecessary), and all members of the cluster are trusted (security is unnecessary). As a result, TIPC proposes a new intra-cluster transport protocol which has been shown to be 35% more efficient than comparable approaches.

The basic TIPC operational unit ("subnetwork") is composed of a full-mesh of individual processors. The next level of aggregation ("zone" or "cluster") is composed of a full-mesh of subnetworks. The highest level of aggregation ("supercluster") is composed of a full-mesh of zones. The hierarchical structure is very similar to Internet subnetwork classifications (Class B, Class C, and so on), however Internet subnetworks are routed using fully-specified destination addresses, and not interconnected via a full-mesh. Within a TIPC cluster, individual processors are uniquely identified by a TIPC address or processor identifier, which is a 32-bit 3-tuple of the form [zone.subnetwork.processor]. The form and representation of the TIPC address are very similar in basic structure to an Internet IP address. In a TIPC cluster, well-behaved applications leverage "location transparency" by referring to service identifiers or logical addresses rather than the physical address of a node or handler. To enable location transparency, all processors within a zone maintain a lookup table to cross-reference between service identifiers and the physical addresses of currently-active handlers for each service (processor + port). Since all processors are interconnected via a full-mesh, changes in the lookup-table are simply broadcast to all other processors. This is similar to the Domain-Name system of the Internet where applications can consistently find a node by referencing its name (i.e., `www.hp.com`) and the domain-name lookup process resolves the specific address. The main differences are that TIPC uses names for "services" rather than "nodes," and changes in the name/address mapping are propagated immediately to all nodes.

4.3.2 The Service Availability Forum (SAF)

The SAF is an industry effort to define open interfaces which will enable carrier-grade platforms, middleware and applications to deliver highly-available services. To facilitate the transition from proprietary systems to standards-based, modular systems, the SAF specifications address open interfaces applicable to Commercial Off-the-Shelf (COTS) devices. SAF goals are to produce specifications which standardize the Hardware-Platform Interface (HPI), the Application Interface (AIS), and the System Management interface.

4.3.2.1 Hardware-Platform Interface Specification (HPI). The SAF HPI describes a standardized "lower" interface between SAF-compliant middleware and the operating system/hardware platform. The SAF HPI draws heavily on the groundwork laid by the Intelligent Platform Management Interface (IPMI) specification to define platform-independent capabilities and data formats. As such, HPI is concerned with the hierarchical description of a system in terms of *resources* and *entities*. In the HPI model, *resources* manage a collection of *entities* by monitoring and recording events pertaining to them and managing their configuration data. *Entities* can be a variety of platform subsystems, such as fans, disks, peripherals, or other manageable units which can expose state information or control functions to the managing *resources*. the HPI specification defines data structures, programming interfaces, and callback semantics for functions to interface with these *entities* using the *resources* that are exposed. For instance, an element in a system might have a sensor to read its current temperature, and a variable-speed fan for cooling it. In this case, an SAF-compliant platform management application would invoke the function `saHpiSensorReadingGet( session, resource, sensor, reading )` to retrieve the current value (`reading`) from a particular temperature sensor (`sensor`) associated with the element (`resource`). If the temperature reading was not desirable, the application could then invoke the function `saHpiControlStateSet( session, resource, control, state )` to change the speed (`state`) of a particular cooling fan (`resource` and `control`) associated with the element.

4.3.2.2 Application Interface Specification (AIS). The SAF AIS describes a standardized "upper" interface between SAF-compliant middleware and highly-available applications. This interface includes an Availability Management Framework (AMF), a software entity separate from the operating system of each clustered node, that coordinates redundant resources within a cluster. In this fashion, the AMF provides a limited "single-system image" view of one logical cluster comprised of several distributed nodes. To effect this coordination function, the AMF determines the state of an application by invoking standardized callback functions and publishes state variables to registered entities. The "higher layer" functions of the AMF are based on reliable cluster infrastructure services such as Cluster Membership (information about cluster nodes), Checkpointing (cluster-wide data objects), Messaging and Event Notification (asynchronous publish/subscribe communication), and Distributed Locking (synchronization facility).

The AIS specification defines data structures, programming interfaces, and callback semantics for AMF functions as well as the various cluster infrastructure services. For example, an SAF-compliant application might invoke the function `saClmClusterNodeGet( node, timeout, info )` to retrieve informa-

tion (`info`) about a node in the cluster (`node`). By invoking this function with `node = SA_CLM_LOCAL_NODE_ID`, a special pre-defined value for the node of interest, the application obtains information specific to the local physical node on which it is executing. The node information (`info`) is returned by the AMF Cluster Membership Service into an SAF-defined memory object (structure). This structure contains basic information such as unique node identifier, node name, cluster name, membership status, communication address, and last boot-time of the node.

5. Commodity Computing Elements

The control plane of the global telecommunications infrastructure is a voracious consumer of compute cycles. It is not surprising that some of the very first digital computers were in fact the control processors of the first large digital voice circuit switches. Nor is it surprising that the telecommunications infrastructure has continuously been one of the principal vertical server markets.

The telecommunications infrastructure requires much more than simple computes. The fundamental importance of human communications has resulted in the market demand for systems that are "always" available, 7×24, at least 99.999% of the time. This equates to no more than 2 minutes of downtime for these systems per year. In addition these systems must continue to operate for a longer period of time than normal commercial computers under catastrophic conditions such as power outages, air conditioning failures, earthquakes, fire and so forth.

These requirements originally led most of the major Network Equipment Providers (NEPs) to design their own compute elements using strict equipment practices. The original switches developed in the 1970s, such as the AT&T 4ESS and the Ericsson AXE were among the worlds first commercially deployed digital computers based on the radically disruptive integrated circuit technology. Consequently the central processors controlling the switch were completely custom designed and, in these examples, adopted an instruction lockstep hardware fault tolerance strategy (HWFT) to achieve the required availability. Other systems such as the Siemens EWSD and the Alcatel System 12 chose to provide the availability through a clustered design. As commercial microprocessors emerged, many of these designs abandoned the proprietary processors and incorporated the commercial microprocessors into the NEP-designed compute elements.

The cost of adding features into the digital switch infrastructure spawned the advent of the Intelligent Network (IN) in the 1980s that introduced an entirely new distributed compute model to the telecommunications infrastructure. Service Control Points (SCP), switching/routing nodes (STP, SSP), and Intelligent Peripherals

created an explosive demand for highly available compute elements. The distributed IN model generalized naturally to accommodate the requirements of wireless telecommunications with the introduction of Home Location Registers (HLRs), Visitor Location Registers (VLRs), Base Station Controllers (BSC), and other network subsystems.

The scope and growth rate of this vertical market for high availability telecom compute elements caused major computer companies such as Tandem, Sun, Stratus, Sequia and Digital to introduce HWFT and clustered telecom servers with equipment building practices to address these markets in the late 1980s and 1990s. Thus, the commercial computer vendor's proprietary server architecture now dominates the control plane of the circuit switched IN infrastructure as well as that of the emerging IP network. Each of these commercial server offerings had a distinct processor architecture, proprietary fault tolerant hardware, proprietary UNIXs and other OSs, as well as high-availability (HA) and management middleware for software fault-tolerance (SWFT).

The Tandem S5000 is outlined in the following section as an example of such an implementation. The systems produced by other vendors were correspondingly unique approaches to addressing the requirements of the "off-switch" telecom market. In this era, like the NEP processors that preceded them, very little standardization existed in the overall architecture of these telecom servers.

During the 1990s a disruptive technology has been steadily gaining force in the commercial server space, independent of any telecom applications. The Personal Computer (PC) commoditization model completely revolutionized the low end processor space in the 1980s. In a natural progression, key elements of these technologies have been steadily encroaching on the proprietary telecom server market. This commoditization model is based on the Intel IA64 and IA32 processor architectures, and is implemented in standard volume servers and a "standard" Linux OS that has been discussed. This Industry Standard model has already almost completely dominated the 32-bit processor architectures, and the current prognosis is that, in the not too distant future, it will also dominate the 64-bit processor architecture. As a result, proprietary server hardware architectures will be displaced or marginalized by the wave of low-cost, high-performance commodity technology. This is demonstrated very clearly by changes in the landscape of server vendors. HP, Compaq, DEC and Tandem representing the high end processor architectures of PA-RISC, Alpha and MIPS have been aggregated into a single company with a strategic high-end vision based on the Intel IA64 architecture. IBM also has a stated commitment to IA64, albeit in addition to its own proprietary Power4 architecture. Only Sun is currently continuing to maintain the proprietary model of the 1980s whereby it produces the central processor, server architecture, operating system, and associated middleware for high end platforms.

The corresponding commoditization of telecom computing elements is enabled by this disruptive technology in the commercial server space. This commoditization is driven by the need for highly cost-competitive telecom infrastructures. The recent, dramatic economic downturn in the global telecom market has accelerated many of these process and business optimizations. Clearly, commoditization of server hardware and software architectures is set to cause yet another far-reaching disruption of the telecom control plane. This commoditized model is based on Intel architecture processors, Linux (with Carrier Grade Extensions) and standard High Availability Software with interfaces such as those specified by the Service Availability Forum (SAF).

To fully understand the radical simplifications that the standards-based model is driving into the telecom server, it is interesting to pick an example of a proprietary telecom server and describe its elements. An interesting case to discuss is the S5000 as it is representative of both a HWFT and clustered architecture. Then, the standard server is outlined in the current two canonical approaches that are being adopted. One is a rack-mounted server that uses only standard communication and storage interfaces, and physically requires only conformance to the NEBS2000 standard (i.e., 19″ rack with 24″ depth with 36U vertical dimension). This architecture highly leverages the implementation of the commercial server as well as commoditized, low-cost components. The other approach is that of a bladed environment. In this area, the Advanced Telecom Computing Architecture (ATCA) is perhaps the leading standards based contender.[6] Various NEPs and telecom server vendors have proposed alternate, pseudo-proprietary architectures. These approaches require a much more detailed physical and interface specification if chassis, blades and other elements from various vendors are to successfully interoperate. Both of these cases are surveyed.

Finally, the standardization of the host processors, general purpose processors, network processors, and special-purpose processors are briefly discussed. It is clear that the Intel IA32 architecture now completely dominates the realm of 32-bit general purpose host processors, and the Itanium family may do the same in the realm of 64-bit processors. Other special-purpose processors are still in the phase of proprietary implementations but over time these architectures may also become increasingly commoditized.

[6] ATCA is a specification for next-generation telecommunications equipment, with a new form factor and based on switched fabric architectures. ATCA has been adopted by the PCI Industrial Computer Manufacturers Group (PICMG), an industry consortium which develops open specifications for high performance telecommunications and industrial computing applications.

5.1 Proprietary Systems

As noted above, the first major outsourcing of control plane elements was to proprietary computer vendor platforms tailored for the telecom market. Examples of such platforms are:

- DEC VaxFT (Vax processor, HWFT, NEBS, VMS), TS series (Alpha 64-bit processor, Tru64 Unix, NEBS, TruClusters HA clustering),
- HP (PA-RISC 64-bit processor, HP-UX, NEBS),
- Stratus (68000, HWFT, NEBS),
- Sun Netra (Sparc processor, HW and non-HWFT Versions, NEBS, Solaris),
- Tandem S Series Integrity (MIPS processor, HW or non-HWFT version, NEBS, NonStop UX Unix, DNP Clustering), Himalaya S Series (MIPS processor, NSK, intrinsically clustered with SWFT check pointing).

Each of these telecom implementations was an original proprietary design. As noted above, many were HWFT with redundant processors running the same instruction stream in order to make the failure of one of the processors essentially transparent to the application.

To understand further the attributes of this generation of telecom servers, it is illustrative to look more closely at the Tandem S5000CO system. The system is illustrated in Fig. 12. The interesting points are the following:

- The system requires a full 36U NEBS frame. For this reason, the individual elements of the systems, Processors, IO cards, power bulks, disks and so forth must enable hot plug, online service of these individual elements, called Customer Replaceable Units (CRUs).
- The system is what we would now call a "blade" system with dual redundant serial ServerNet System Area Network (SAN)—"a network with bus semantics"—on the backplanes. This SAN is a precursor of the standardized Infiniband SAN and influenced its early development. The radial interconnect of ServerNet facilitated hot-plug and fault isolation while providing a redundant communications bus. Each CPU and IO controller has a connection into each of the dual redundant fabrics.
- The system has dedicated redundant service processors that provide system console functions as well as monitoring and control capability for various power, thermal and fan functions.
- Each of the host processors is capable of operating in a 4-way symmetric multiprocessing (SMP) configuration. Each processor board is designed with redundant processors on the board running in lockstep and checking each other for

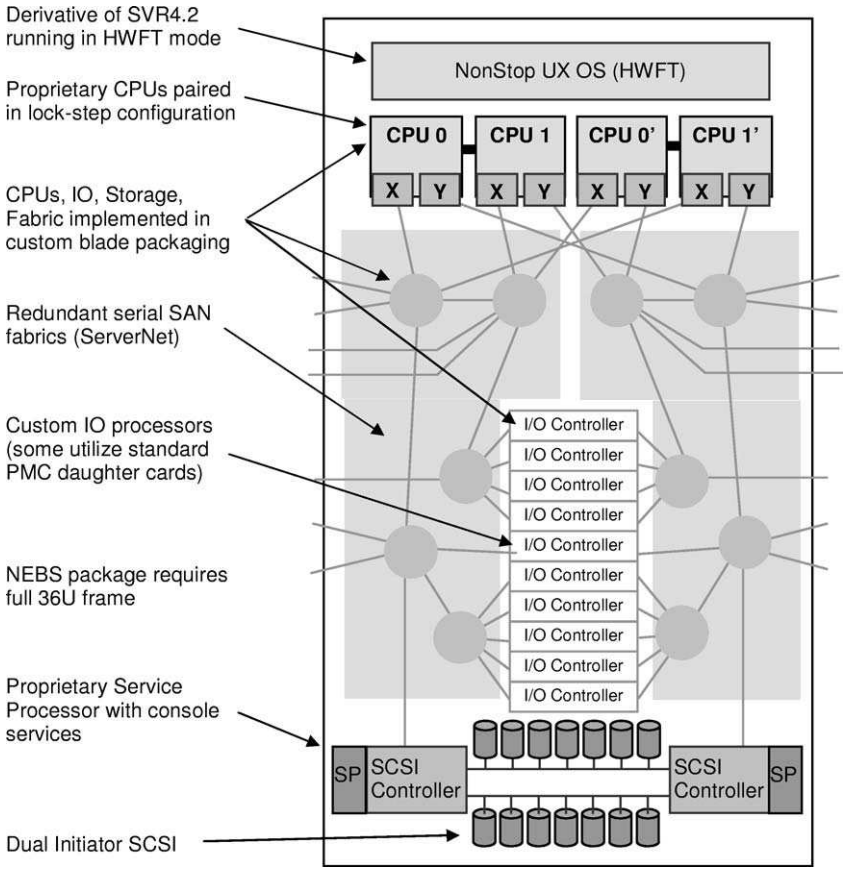


FIG. 12. This figure illustrates the S5000, an example of a proprietary HA telecom server. Typical of systems of this generation, the design is a completely proprietary blade system and shown in a HWFT mode. This system also supports a SW HA mode with proprietary HA middleware as a cluster over the ServerNet fabric.

failures. This is likewise true of the ASIC support chipset. Memory is protected by ECC that is encoded and decoded in the self checked domain. The IO from the board is through ServerNet that is redundant with numerous self-checking features.

- The CPU boards can optionally be paired into a HWFT configuration where each separately-serviceable CPU board is running the same instruction stream. This enables failures of an entire SMP processor complex to be transparent to

the application. The Service Processor and system firmware have the capability of introducing a new or repaired board into service with only sub-second disruption at the application level for 2 GB memory images.

- The CPU boards can also optionally be run in a simplex mode with single-system image clustering and SWFT using HA clustering software.
- The OS was a proprietary derivative of UNIX SVR4.2 that incorporated multiple hardening features. The operating system was also modified to support the platform's HWFT features, including custom drivers to enable use of ServerNet for inter-process communication as well as external I/O.

5.2 Industry Standard Systems

Two variations of the industry standard telecom server are emerging that promise to homogenize telecom network elements around a standard Linux/Intel architecture. One version is the rack mount server that highly leverages the standard commercial servers into the telecom space. The other is the blade server for which there is significant standards based (PICMG) support around the PICMG 3.x, or ATCA. Each of these platforms optimizes different physical implementation criteria but presents a common Linux/Intel programming and operating environment. A diagram of the industry-standard server is shown in Fig. 13.

5.2.1 Rack-Mount Servers

Telecom-grade rack mount servers take advantage of the fact that many two-CPU-socket, commercial-grade servers can be readily repackaged into a NEBS compliant form factor. This repackaging can often be accomplished without modifications to the logical architecture or the physical implementation of the main CPU boards, PCI IO subsystem, and storage facilities. This fortunate situation is a by-product of the fact that two-socket commercial-grade servers using either the IA32 or IA64 processors are typically manufactured in 28–32″ deep boxes (for an 800 mm rack). In this configuration, the components are almost universally arranged with the back third of the box having the main CPU board and PCI IO cage, with connectors on the back of the box, and with storage and bulk power in the front third of the box with access from the front. Typically, simple cabling is used to traverse between these regions for power distribution and peripheral connections.

To leverage this design into a NEBS compliant telecom server, the rack depth must conform to the 600 mm NEBS2000 rack depth standard. This proves to be one of the primary problems; however, after replacing the commercial-grade AC power supplies with telecom-grade DC supplies, the front third of the box can be "folded under" so that the telecom-grade form-factor actually occupies more height in the

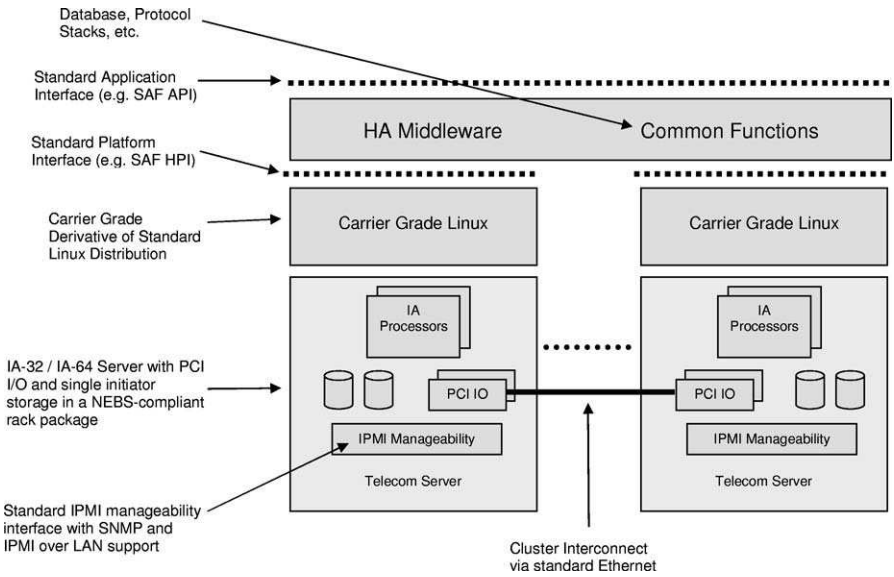


FIG. 13. Illustration of the industry-standard telecom server.

rack. This physical modification also happens to improve the cooling flow, which is a significant issue for telecom-grade equipment. These modified units are typically 1–4U in rack height and have moderate weight so that they can be easily serviced by a single individual. With redundant fans and (hot plug) redundant disks and bulk power, the effective reliability of this complex, with a compliment of PCI IO, is comparable to a CPU-only CRU in the proprietary designs. Multi-node configurations interconnected by Ethernet, Infiniband, or other switch fabric can be clustered into highly-available systems. These HA features take advantage of the standard Intelligent Platform Management Interface (IPMI) subsystem running on the onboard "service processor" to monitor various aspects of system health, including power, fan and thermal conditions. In this model, each simplex server box can be considered as a "CRU" with power bulks and disks as optional sub-CRUs due to their comparatively high failure rates.

Notice that in this discussion of commodity computing elements, there is no mention of high data integrity lockstepping between CPUs. Likewise, there is no mention of HWFT duplex features. The industry has essentially concluded that the level of integration in these devices (and hence increased reliability) coupled with error control features (including extended chip spare ECCs protecting against the failure of an entire memory chip) has eliminated the need for the legacy HWFT features in

this class of computer for telecom applications. Of course, there are still classes of applications such as major financial transactions where data integrity remains an absolutely paramount requirement. Almost all the proprietary hardware FT designs are giving way to software HA designs, driven by the realization that the preponderance of the faults are now in the software, not the hardware.

This has major advantages in leveraging all the high technology components, hardware and firmware and associated supply chain infrastructure to enable very cost effective designs. These designs also benefit from the reliability improvements that a large deployed base drives aggressively. In addition, these systems can leverage the highly aggressive performance curve (including $\times 2$ or greater SMP designs per processor socket) that a unique telecom design could not support. Clearly, this needs to be balanced against the need for longer lifetimes and less churn that the telecom market requires.

5.2.2 Blade Servers

Blade servers are a more direct generalization of the conventional telephony systems model where a multipurpose chassis can contain several function-specific cards for processing or I/O. As an example, the PICMG 2.x Compact PCI (cPCI) standard for telecom environments enabled up to 21 6U high cards to be hot-inserted into a backplane in an enclosure with supporting redundant power bulks and fans. Typical systems are about 9–12U in height, including the power and cooling facilities. CPCI systems support a PCI multi-drop bus on the backplane, with hot-plug extensions. There are very difficult fault isolation problems with the PCI multi-drop bus and as the bus evolved to PCI-X at 100 MHz or 133 MHz the number of supported slots dropped to less than 4.

To compensate for these bus problems, the PICMG 2.16 extension added redundant Ethernet in the backplane as a radial interconnect with two of the slots designated for use as Ethernet switches. While this solved many of the fault isolation and PCI-X bus scaling problems, the 6U form factor and associated card pitch, power and thermal characteristics became problematic with the newer generation of 32-bit processors and prohibitive for the IA64 processors.

As a result, the PICMG 3.x ATCA blade has been proposed with a larger form-factor: 8U in height and 280 mm deep with a $1.2''$ pitch. This pitch supports standard DIMMs (dual in-line memory modules) and provides more heatsink room for processors and other hot chips. The ATCA form factor can easily support $\times 2$ or greater SMP Pentium-4 systems as well as one- or two-socket Itanium solutions, although in the latter case possibly requiring two card slots. An ATCA chassis supports 14 slots in a standard $19''$ rack. Rather than supporting PCI-X buses on the backplane, ATCA supports the radial and fully interconnected mesh backplanes for Ethernet,

Infiniband and other serial interconnects such as PCIExpress. Including power and cooling enclosure, a typical 14 slot ATCA system is expected to require about 13–16U of vertical space in a rack.

ATCA servers represent an industry standard evolution of legacy proprietary blade architectures such as those pioneered by systems like the S5000. In the industry standard system, the processor technology is driven by Intel and it is expected that multiple vendors will provide inter-operable chassis and blade elements. In addition to the general purpose processor blades, blade offerings are likely to include specialized functions such as dedicated network processors and DSP processors as well as IO cards for network and storage connectivity. Although the blade system architecture is very compelling for dense computation environments and can be configured for highly-available purposes, it is unlikely that these elements will be used to implement HWFT systems such as lockstep CPU architectures. Additionally, as with the industry-standard rack mount servers, the HA strategy for such systems will rely on cluster software which interacts with IPMI and OS resources to determine the health of a node.

5.2.3 Challenges—Standards, Interoperability, Investment

The primary advantage of the blade strategy is that it provides a dense mix-and-match collection of general purpose and special purpose processing elements in addition to IO and media interfaces. These standalone elements can then easily share resources via the substantial bandwidth provided by the mesh interconnect on the backplane. This becomes very important when the outsourced control and dataplane elements are implemented in a single infrastructure in the converged network model. This converged model could prove to be yet another disruptive technology beyond the standardization of these elements as the evolution of the network drives to a truly IP based infrastructure.

However, in this picture of the blade strategy, we have failed to mention that there are several other "standards" competing with ATCA. These architectures are being proposed by some of the major NEPs and by telecom server vendors. This is causing the blade market to be fragmented and fraught with ambiguity. As with any "industry standard," and with the status of blade architectures in the telecommunications space, this is the challenge: Everyone has to buy into the standard for it to achieve the desired cost and interoperability advantages.

Even with broad acceptance of a single standard, the blade strategy, whether ATCA or otherwise, is less able to directly leverage the economies of scale of hardware from the commercial server space. The primary logic boards, processor, management, and switch must be developed directly for the ATCA implementation and cannot simply be leveraged. Furthermore, the ATCA standard has many configurable

options, starting with the choice of the fabric: star or mesh, Ethernet or Infiniband, and a host of other variants. These configurations require many combinatorial possibilities for variations of processors, IO blades, and so on. The net effect is to further fragment the market for a given blade configuration. While the few "best" configurations may eventually emerge from a Darwinian "natural selection," this process will take some time and will create a barrier-to-entry for development. Development of these cards for this class of technology requires a substantial investment and, with the fragmented state of blade standards, it is not clear whether multiple vendors can sustain a viable business in the resulting market.

Thus, the definition of a standard telecom server blade is far from complete. In contrast, the rack-mount server approach has very simple interoperability requirements between vendors because the enclosure and interface standards are relatively mature. These considerations simply require that the server (a) fits within the NEBS2000 enclosure definition, which is compatible with the standard 19″ rack, and (b) uses standard communication and storage interfaces such as Ethernet, SCSI etc.

If the telecom blade definition finally consolidates on a single standard, it could set the stage for the next disruptive technology of the fully converged network where data-plane and control-plane functions are fully consolidated onto a single common infrastructure.

5.3 Industry Standard Processors

The Intel Architecture completely dominates the industry volume for 32-bit general purpose host processors, and is supported by most of the major computer and server vendors. By providing processors to multiple system vendors, Intel and AMD are able to garner enough market share to support the substantial research, development, and manufacturing costs required to remain competitive in the standard processor market. The recent years have seen the performance of these processors increase exponentially to the multi-GHz clock rates currently available.

In telecommunications applications, high-speed, large-scale databases and high throughput for asynchronous events are critical features for network elements. To support these applications, processors with 64-bit computation and address-space have significant performance advantages. To date, the high end 64-bit processor market has been focused on proprietary processors based on the Reduced Instruction Set Computer (RISC) architecture. Proprietary architectures include the DEC Alpha, HP PA RISC, IBM Power4, SGI MIPS (and Tandem), and Sun Sparc. However, Intel, in collaboration with HP, have introduced the IA-64 architecture with the goal of driving the highly successful 32-bit industry-standard business model into the 64-bit space. This objective may have been accelerated by the independent corporate consolidation that collapsed the DEC Alpha, PA RISC and a major application of

the MIPS processor onto the IA-64 architecture. AMD is making a similar play with the Opteron architecture. The remaining proprietary competition in 64-bit processors are systems based on the Sparc and the Power4, although IBM is also introducing systems based on IA-64. The IA-64/Itanium architecture is very different from the IA-32 approach and IA-32 code is not natively executable by Itanium processors. In contrast, the Opteron processor approach is more evolutionary with a stated goal to run IA-32 code natively very efficiently while providing the extensions for native 64-bit operation. Regardless, both of these architectures are driving the industry-standard host processor model into high-end 64-bit systems. Computing elements for telecommunications may be among the first adopters of this "disruptive" technology. Following sections briefly review competing industry-standard 64-bit architectures in the context of telecommunications systems.

5.3.1 IA-64 and Itanium

The IA-64 architecture is a completely new processor design based on the Explicitly Parallel Instruction Computing (EPIC) model. This model more effectively utilizes processor resources and minimizes silicon complexity while supporting multiple instructions per cycle. Experience with current processors shows that the processor resources are idle a substantial percentage of the time waiting on data (from caches, main memory, etc.) and waiting for branch decisions to be made. The EPIC approach enables a much higher degree of speculative execution to occur (utilizing resources that otherwise would be idle) while keeping the complexity manageable. The key to this optimization is in making the parallelism explicit in the architecture and pushing the complexity into the compile stage, where more resources (time, memory, etc.) are available and better optimization decisions can be made. With this approach, the processor can realize a high rate of instructions-per-cycle on real world workloads while enabling higher frequency designs than the lower complexity (relative to similar CPU designs without EPIC) would afford. The EPIC designs also tolerate greater mismatches between the memory and processor speeds due to the speculative prefetch capabilities which may become more important as the processor clocks are driven even higher.

With its EPIC architecture, an IA-64 system will allow concurrent execution of multiple "bundles" of 41-bit instructions, 3 instructions per bundle. To maximize efficiency, the compiler attaches a header to each bundle indicating which CPU subsystems are required by the instructions. The IA-64 architecture maintains 128 64-bit Integer Registers, with an extra 65th bit per-register for speculative operations. Many of these registers are reserved for stack manipulations during function calls. IA-64 also has 128 82-bit Floating Point Registers divided into partitions that can be independently disabled (masked for access faults) to make some context switching more

efficient. Multiple Application Registers (AR) are available to hold stack state, enable atomic and loop operations, assist in IA-32 emulation, and so on. Some ARs are also "kernel registers" with asymmetric unprivileged-read/privileged-write modes, useful for posting non-sensitive kernel state for access by non-kernel processes. Additionally, IA-64 uses 64 1-bit Predication Registers to minimize the occurrence of indeterminate state associated with branch conditions. By prefixing conditional instructions with partial comparison results, complex branching logic can be executed with less ambiguity and minimal clock-cycles [44].

IA-64 also leverages the EPIC architecture to manage 4 types of interrupts: *Abort* (internal hardware-triggered), *Interrupt* (external, asynchronous events triggered by peripherals, 16 priority classes of 16 vectors each), *Fault* (execution errors), and *Trap* (operations requiring software assist). *Interrupts*, *Faults*, and *Traps* are handled by an Interrupt Vector Table initialized by the operating system. This table contains Interrupt Service Routines with up to 64 *bundles* of instructions. As a result, many common interrupt scenarios can be handled inside the table without further branching.

The Itanium family of processors, which debuted in mid-2002, are a "realization" of the IA-64 "theory." As such, Itanium is a 64-bit processor with *adequate* 32-bit performance. The IA-32 Execution Layer (IA-32 EL) is a software layer to help handle 32-bit applications on the third-generation Itanium processor at speeds equivalent to a 1.5 GHz Xeon MP. In this case, the software emulation imposes a significant performance penalty over the native hardware clock speed.

The early-generation Itanium chips also have some important differences with respect to the IA-64 architecture. In the IA-64 model, the processor accesses a flat 64-bit address space, with some caveats to handle efficiency challenges of a single large, sparse region. The upper 3-bits of the 64-bit address are used to select a "virtual region" via special 24-bit registers which point at unique regions of memory. So, at any time a program can access up to 8 of the 2^{24} regions, each of which contains 2^{61} bytes. In the Itanium implementation, the processor accesses a 44-bit physical address space (with a single-bit cached/uncached indicator) and a 54-bit virtual address space. In this case, the upper 3-bits of the 54-bit virtual address select the "virtual region" via 18-bit registers pointing at unique regions of memory. So, at any time, a program can access up to 8 of the 2^{18} regions, each containing 2^{51} bytes. These virtual regions enable efficient process management for multi-tasking environments, and pages can be protected using more than 16 "permission keys" (so, where the IA-64 model has 24-bit keys, the Itanium processor has 18-bit keys) [44].

5.3.2 Opteron

In contrast with the IA-64 architecture and Itanium implementation, the Opteron processor from AMD is a native 32-bit processor with performance enhancements

for 64-bit operations and close coupling with other processors. In effect, the Opteron extends the legacy 'x86 architecture to be useful with a 64-bit operating system. As a result, 32-bit 'x86 applications can run at the full clock speed of the hardware (for example, a 1.8 GHz Opteron could run applications at speeds similar to an IA-32 2 GHz Xeon/MP). In *Legacy* mode, Opteron can boot directly to DOS or a 32-bit operating system and run 32-bit/16-bit applications just like an IA-32-class processor. In *Long* mode, Opteron can boot into a 64-bit OS where its 64-bit registers and 64-bit memory and disk access will improve data-handling performance significantly over the pure IA-32 architecture. In this mode, Opteron can address 1 TB (1024 GB) of physical memory and 256 TB of virtual memory without losing backward compatibility for 32-bit/16-bit applications [45].

Although the Opteron has separate, relatively slow 64 kB level-1 instruction and data caches and a 1 MB level-2 cache, the integrated memory and bus controllers compensate somewhat for the latencies inherent in its non-uniform memory architecture (NUMA). An interesting feature in the Opteron architecture is the trio of 6.4 GB/s (full-duplex) *HyperTransport* buses to link peripheral devices (other processors, memory, I/O devices, etc.) [46]. Usually in an SMP configuration, NUMA processors have "isolated" memory, which makes cross-processor sharing and memory access relatively slow. Opteron's high-speed *HyperTransport* links mask this effect, more tightly coupling the separate processors [45].

5.3.3 Specialized Processors

Telecom makes considerable use of Digital Signal Processors (DSPs) and more recently network processors in bearer path processing. DSPs have resources such as registers specifically for multiply/accumulate which is common in signal processing (i.e., filtering operations) along with custom opcodes to use them efficiently. Network processors typically have multiple packet-munching FIFO's that can be programmed (much like microcode) to perform repetitive packet-specific functions like header manipulation or payload inspection to offload the host CPU. The application of these specialized processors can range from traditional applications such as μ-law and A-law codecs to the more recent network and compute-intensive security applications such as IPSec. These processors are still further up the proprietary evolution path but their standardization will surely be a significant disruptive technology in the converged network.

Architectural trends in future general purpose processors is likely to include multicore CPU's with several silicon cores in a single package. In addition to requiring less electrical power (an issue in dense telecommunications deployments), multicore CPU packages will enable faster sharing of data between processors (much like the Opteron *HyperTransport* bus), including critical memory segments such as first- and second-level cache.

6. Protocols of Significance

Technology disruption is usually expressed in the form of a product which drastically and suddenly alters the value proposition for a market [1]. In the context of IP-based networking and telecommunications, "disruption" also occurs in the form of important protocols or communication mechanisms that become standardized or widely adopted to the point of defacto standardization. Three such protocols (or, protocol families) are discussed in following sections. These IETF-standardized approaches to Internet networking fundamentals (IPv6), transporting telephony signaling over IP (SIGTRAN and SCTP), and media session control (SIP) are poised to radically affect the structure and use of next-generation telecommunications networks.

6.1 IPv6: The Next Generation Internet

The IP Protocol Version 6 (IPv6, RFC2460–2463, 2402, 2406, 3484, 3513, and others) is not a disruptive technology in itself; rather, it is a well-designed reaction to the many disruptions of the "convergence" process. From the telecommunications perspective, the benefits of IPv6 led to its adoption as the default IP protocol for all third-generation (3G) mobile networks. IPv6 is designed to address critical areas such as IP address depletion, efficient packet handling, authentication and security, and ease of network administration. These benefits can be discussed in terms of three areas where IPv6 characteristics are potentially the most significant: Flexible, Efficient Headers; Hierarchical Addresses; and Host Autoconfiguration.

6.1.1 Flexible, Efficient Headers

The architecture of packet headers in IPv6 is vastly improved from the case of the IPv4 header. This structure leverages the hierarchical IPv6 address space and simplifies the forwarding task of core routers. In IPv6, the base header is a 40-byte fixed-length object with only a few basic fields. This contrasts with the variable-length IPv4 header, with almost twice as many fields. The base header can be augmented with an arbitrary number of optional "extension headers" which describe special routing, handling, or other options for the packet payload.

In addition to its native support for security via the *Authentication* and *Encryption* headers, IPv6 header extensions for packet and session handling may enable some forms of per-stream Quality of Service (QoS). For example, the *Fragmentation* and *Source Routing* headers can be used in end-to-end path optimization. IPv6 nodes must adjust their Maximum Transmission Unit (MTU) to optimize fragmentation on an end-to-end basis (IPv6 routers do not fragment packets). The IPv6 *Fragmentation*

header and MTU discovery process allows for specification of fragmentation boundaries that is transparent to higher-layer applications and "friendly" to the transport network. Based on the MTU discovery process, the optional *Source Routing* header can be used by the transmitting node to explicitly control the path of packets through the network. When used for "strict" forwarding, the *Source Routing* header contains a list of admissible next-hop forwarding nodes that intervening routers must adhere to. The *Destination Options* and *Hop-by-Hop Options* headers can also be used for transmitting management information such as RSVP resource reservation to each forwarding node along the path as well as control or handling information destined only for "special" nodes, including the destination. Using these extension headers, conversing IPv6 nodes can conceivably test and select network paths (or use preconfigured paths) to minimize the latency of multimedia or control streams.

6.1.2 Hierarchical Addresses

As the Internet has grown, the limited, non-hierarchical structure of the IPv4 address space has become problematic. Although Classless Interdomain Routing (CIDR)[7] has compensated for some of the IPv4 addressing weaknesses, the advanced hierarchical address space of IPv6 will facilitate vastly more efficient routing and management architectures. In addition to the well-known 128-bit length of the IPv6 address space—6×10^{23} unique addresses per square meter of the Earth's surface—the most significant characteristic of IPv6 addresses is their "aggregation-based" allocation. With this structured approach, the IPv6 address space has *geography-dependent* and *provider-dependent* components. The result is a separable "trunk," "branch" and "leaf" architecture that allows efficient routing and simple re-attachment of the least-significant bits of the address based on the location and characteristics of the most-significant bits.

Aggregation-based addressing brings the IP address structure more in-line with addressing schemes existing in telecommunications networks, where routing decisions can be made quickly by examining specific segments of the globally-unique network address. The "phone number" is one component of the network address, which also includes "country codes," "area codes," and so on. In addition to aggregation-based addresses, IPv6 allocates certain address ranges for multicasting (similar to IPv4) and for limited-area ("site-local" and "link-local") use. The "link-local" address plays an important role in the autoconfiguration process for IPv6 hosts.

[7] CIDR uses bit-masks to allocate a variable portion of the 32-bit IPv4 address to network, subnet, or host. This technique allows a single prefix address to refer to a collection of small, plentiful Class C networks, emulating the larger scope of sparse Class A and Class B networks.

6.1.3 Host Autoconfiguration

Among the most significant improvements offered by IPv6 is its address autoconfiguration features. Demand for IP-based networking by mobile users is increasing rapidly. IPv6 allows mobile devices to easily, automatically, and scalably transition to new addresses as they move among foreign networks. A fundamental assumption in IPv4 is that each node always has the same point of attachment to the network. As a result, IPv4 addresses essentially identify a static network segment (link) rather than a node, and support for mobile devices is awkward at best. In contrast, IPv6 addresses contain information about the network attachment point ("trunk" and "branch") as well as the node itself ("leaf"). This characteristic is enabled by the *stateless & stateful* address configuration protocols innate to all IPv6 nodes, and it is efficiently exploited by the Mobility extensions for IPv6 (MIPv6). In particular, an IPv6 host can *statelessly* configure its own IPv6 address without human intervention or the help of a network-resident (or, *stateful*) server. To accomplish stateless address configuration, the host observes or requests a valid *address prefix* from its current network connection, then concatenates a globally unique *link-layer address* based on its network adapter identifier. The process is equivalent to dynamically replacing the most-significant bits of the address with a locally-acquired (network-specific) identifier, and deriving the least significant bits of the address from a unique property of the mobile device.

Mobility (and constant connectivity) for IPv6 nodes is enabled by this addressing scheme in concert with their ability to maintain and prioritize multiple active addresses for each network interface. For instance, MIPv6 nodes are always identified by their (static) home address, and packets sent to this address are tunneled to the mobile node in a fashion similar to MIPv4. However, while in a foreign network, MIPv6 nodes can use stateless autoconfiguration to determine a new, globally unique *care-of-address* that leverages the hierarchical address structure. Then, the mobile node can use both addresses to effect a form of automatic packet re-routing. A key component in this process is the binding cache kept by all IPv6 nodes. The binding cache is a lookup table between known, static addresses and temporary, care-of-addresses. Each packet transmitted by an IPv6 node is cross-referenced against the binding cache, and if a care-of-address is found, an additional routing header is inserted into the packet. The effect of this process is route optimization for the "triangle routing" problem common in mobile IP scenarios. Via binding cache updates, IPv6 nodes are able to track the network position of other nodes and efficiently re-route transmitted packets *without* inefficiencies of IP-in-IP tunneling.

6.2 SCTP: The Commoditization of SS7

Many desirable features of modern telecommunications depend on efficient, robust signaling between elements in the network. By physically separating the "control channel" from the "data channel," operators of circuit-switched networks have been able to optimize the performance of both network components for the existing telephony paradigm. The primary framework for the "control channel" in circuit-switched telecommunications is Signaling System #7 (SS7). Many of the performance benefits derived from SS7 are due directly to the separate optimization of control and data paths. However, motivated by projections of greater efficiency (meaning "economic advantage"), the trend among next generation network operators is to combine network control and data channels for transport through a packet-switched cloud. In this approach, control as well as data packets are exchanged over a common core network, typically based on the Internet Protocol suite.

In the end-to-end paradigm of the Internet, the core network is essentially a "stupid bitpipe" and all complexity is "pushed to the edge." This requires a drastic re-implementation of the error control, flow control, and management mechanisms of conventional telephony that had previously been embedded in the SS7 network elements. The architectural concepts for this re-implementation are described in RFC 2719, *Framework Architecture for Signaling Transport*. The Stream Control Transmission Protocol (SCTP) is a fundamental outcome of these efforts. The purpose of SCTP is to approximate the robustness of telephony's separate SS7 control channel while exploiting the highly desirable flexibility of the Internet's common, packet-switched data channel.

The SIGTRAN architecture and SCTP adaptation layers seem grossly overcomplicated at first look. However, emulation of multiple types of SS7 functionality requires a certain level of intricacy. In general, an adaptation layer mediates between application-specific SS7 requirements for a particular SS7 user and the underlying SCTP transport. As part of this mediation, the "xUA" must expose the API and functionality toward the application (upper layer) that it expects from a "real" SS7 node. Further, the "xUA" may have to rely on external SS7-based nodes for some things that aren't present locally on an IP-based node.

In addition to carrying various SS7-user messages over IP, SCTP is a general transport protocol, and is capable of broader applications. SCTP has been proposed for use in several applications, including as transport between SIP entities, and in conjunction with various security and authentication mechanisms, such as IPsec or DIAMETER.

6.2.1 Key Concepts in SCTP

The Stream Control Transmission Protocol (SCTP) is designed to transport PSTN signaling messages over IP networks. RFC 2960 describes the following key points of SCTP along with the explicit definition of packet and header fields, required attributes, interface between SCTP and the "upper layer protocol," and other protocol details.

- *Endpoint*: An SCTP *endpoint* exists at either end of an SCTP session. Usually, the endpoint is defined by the data structures, socket instances, and other state information which is specific to SCTP. This information is managed by the operating system of the host computer, and exists between the SCTP user and the IP network. The "SCTP user" is typically an adaptation layer for higher-level SS7 entities. Several virtual endpoints can exist per logical node, in the same way that several IP sockets of the same type can be created on a host. More specifically, an SCTP endpoint is a collection of *transport addresses* using a common IP port number, along with the information needed to manage the connections. In object-oriented terms, an endpoint is an instance of the class "SCTP" together with its data and methods.
- *Association*: The basic service offered by SCTP is the reliable transfer of messages between peer SCTP users. It performs this service in the context of an *association* between two SCTP endpoints. Once the association is established, unidirectional streams are open for data transfer on both ends. Figure 14 describes the initialization process for an SCTP association. In the figure, note the use of various types of "chunks" used in session establishment as well as the negotiation of other communication parameters. An association must be created between two SCTP endpoints before they can transfer data. Assocations can be negotiated explicitly via an `ASSOCIATE` request from an SCTP user at one endpoint or implicitly via data transmission with default association parameters. During initialization, a state-cookie mechanism is used to provide some limited security. This is illustrated in Fig. 14. Note that no more than one association is allowed between two SCTP endpoints, and the association is identified by the IP port number to which it is bound. After initialization, the association assigns a global Transmission Sequence Number (TSN) to packets as they are transmitted. The TSN is independent of any per-stream sequencing. The association also keeps track of idle destinations using a heartbeat message which must be acknowledged within a certain time-period by the receiver via a `Heartbeat-ACK` chunk.
- *Transport address*: Each SCTP endpoint provides the other endpoint with a list of transport addresses when the association is created. An SCTP endpoint can

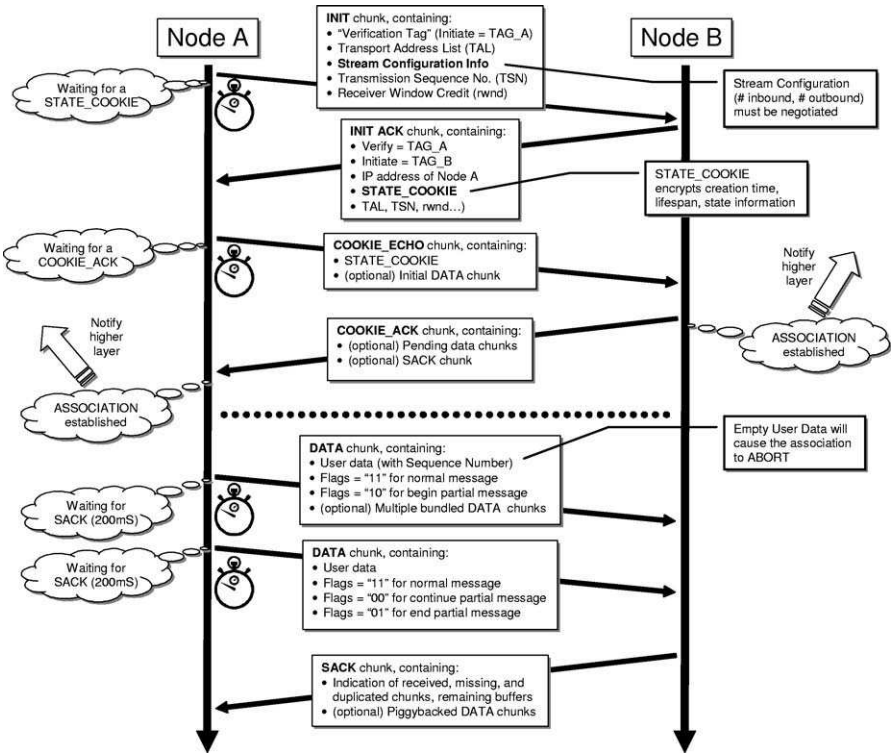


FIG. 14. Association initialization in SCTP.

transmit or receive IP packets from any of the transport addresses that it announces. A transport address is unique to an SCTP endpoint, and is comprised of an IP address in combination with an SCTP port (i.e., an Internet "socket" connection). All transport addresses used by an SCTP endpoint must use the same port number, but can use multiple IP addresses for failover or parallel transmission. This is also known as *multihoming*, and is described in detail below.

- *Streams*: An *SCTP stream* is a sequence of user messages ("chunks") to be delivered, in correct sequence, to the upper-layer protocol. This contrasts with the *TCP stream* which is an ordered sequence of bytes. The number of streams in an SCTP association is negotiated at startup, and a primary path is designated. Streams are somewhat analogous to threads in process execution, or "virtual

channels" in other telecommunications parlance. Streams are numbered, and have individual 16-bit sequence numbers to ensure ordered transmission.

- *Packets and chunks*: An SCTP packet is comprised of a header followed by "chunks" of data. A chunk is a unit of information consisting of a chunk-specific content, header, and format. Chunks can be `CONTROL` chunks or user data encapsulated within `DATA` chunks, and may be bundled (multiplexed) in a single SCTP packet. Each chunk is identified by an 8-bit type-tag in the chunk header.

6.2.2 Path Management

A significant feature of SCTP is its explicit approach to path optimization. SCTP performs this optimization between destinations independently of any transport or routing facilities provided by the IP network. The interesting path management features of SCTP can be divided into two primary areas: Multi-homing and Congestion Avoidance.

Multi-homed IP nodes are nodes which can be reached via several different IP addresses. If an SCTP endpoint is multi-homed, the SCTP instance informs corresponding nodes about all of its IP addresses during association setup (using the `INIT` chunk's address parameters). An SCTP instance regards each IP address of its peer as one "transmission path" towards this endpoint.

SCTP is unique in its use of multi-homed capabilities because it allows the SCTP instance on an endpoint to monitor the relative quality of the various paths in the association and select the "best path" for transmission. This essentially allows the SCTP instance to participate in routing decisions. As a result, if the IP network(s) connecting two multi-homed SCTP endpoints are configured so that traffic destined for different IP numbers travels on physically separate paths, then the SCTP association becomes somewhat tolerant to network failures. Multihoming can also be useful in optimizing throughput and avoiding congestion.

As in TCP, SCTP uses two modes for congestion avoidance, called *Slow Start* and *Congestion Avoidance*. Transition between these modes is determined by a set of path-specific variables which are dynamically updated based on path performance. The association begins with the endpoints in *Slow Start* mode. When data is successfully delivered and acknowledged, the Congestion Window variable (`CWND`) is increased. Eventually, `CWND` exceeds the *Slow Start Threshold* (`SSTHRESH`) and the SCTP instance transitions into *Congestion Avoidance* mode. While in *Slow Start* mode, `CWND` is generally increased quickly (roughly one MTU per received `SACK` chunk), whereas in *Congestion Avoidance* mode, `CWND` increases by roughly one MTU per Round Trip Time (RTT). Transmission timeouts or other events that trigger retransmission cause the `SSTHRESH` variable to be reduced in addition to resetting the `CWND` variable. For example, a timeout event causes the SCTP instance to tran-

sition back into *Slow Start* mode with `CWND = MTU`, and a Fast Retransmit event sets `CWND = SSTHRESH`.

Note that since the Maximum Transmission Unit for each path (Path MTU) is directly related to congestion control, an SCTP instance determines the MTU and maintains current values of this quantity for each path. Transmissions are also fragmented according to the MTU for each path.

6.2.3 Adaptation Layers

The transport of SS7 signaling over an IP network requires the use of applications that exist at various layers in the SS7 protocol stack. These applications will need to run without modification on IP-based nodes having no SS7 lower layers. As a result, the IP-based nodes will have to act like native SS7 nodes even when they may not (locally) have access to that information or those capabilities. Note that in SS7, each layer is a *user* of the supporting layers beneath it. As a result, upper layers make assumptions about information the lower layers should be able to provide. This includes information about the network architecture, conditions, paths, etc. as well as information about the corresponding process at the other end of the channel. To reproduce this functionality at a specific layer boundary, this behavior must be completely emulated, and must maintain access to the same capabilities with the same programmatic interfaces, return values, and other data.

To achieve these performance requirements, the IETF SIGTRAN working group has defined a collection of *adaptation layers* for SCTP which emulate specific functionalities found in the various layers of the SS7 stack. Several of the SIGTRAN adaptation layers are described in following sections.

- *M3UA*: The MTP3-User Adaptation Layer (M3UA, RFC 3332) supports the transport of MTP3-User signaling (e.g., messages from ISUP, SCCP, TUP, etc.) over IP using SCTP. To achieve seamless operation of peer MTP3-Users in separate IP and SS7 domains, M3UA provides a functional inter-working of transport functions. This allows an SS7-based MTP3-User (such as found on a signaling gateway) to communicate "normally" with an IP-based Application Server Process, such as a database or media gateway controller. From an MTP3-User's perspective, M3UA appears to be MTP3. It responds in the same fashion to queries and produces the same information about the network or far-end application. In effect, M3UA extends the MTP3 services of an SS7 node to remote IP-based applications. M3UA does not itself provide the MTP3 services. Instead, it uses SCTP/IP for transport to access MTP3 services at a remote node. In this fashion, an IP-resident MTP3-User is unaware that the expected MTP3 services are provided by an MTP3 Layer at a remote signaling gateway, and not by a local MTP3 layer. The MTP3 layer at the signaling gateway may

also be unaware that its users are actually remote user parts backhauled over IP using M3UA/SCTP. M3UA can also be used for point-to-point connections between IP-resident MTP3-Users. In this case, M3UA emulates a subset of MTP3 services sufficient to manage the connection transparently.

- *M2UA*: The MTP2-User Adaptation Layer (M2UA, RFC 3331) is responsible for "backhauling" signaling messages from an MTP2-user (such as MTP3) over IP using SCTP. As such, M2UA is a remote procedure call mechanism which uses SCTP for the transport protocol. M2UA allows IP-based nodes without a local MTP2 implementation to use MTP2 facilities on a remote node. Using SCTP as its transport across the IP network, M2UA masquerades the remote MTP2 as a local facility. For example, from the perspective of an MTP3 layer on an IP-based node, M2UA is MTP2, but it uses SCTP/IP for transport rather than SS7, and the MTP2 facilities are actually provided by a remote node.
- *M2PA*: The MTP2 Peer-to-Peer Adaptation Layer (M2PA) [47] adapts MTP3 signaling messages for transport over IP using SCTP. Using SCTP for transport, M2PA masquerades MTP2 between SS7 nodes that have IP connectivity. In other words, an IP-based node with upper-layer SS7 application processes can function as a traditional SS7 node using the IP network instead of SS7 links. To accomplish this, the SCTP association acts as an SS7 link between the communicating endpoints. This allows for full MTP3 message handling and network management capabilities between any two SS7 nodes communicating over an IP network.
- *SUA*: The SCCP-User Adaptation Layer (SUA) [48] supports the transport of SCCP-User signaling (e.g., MAP and CAP over TCAP, RANAP, etc.) over IP using SCTP. When SS7 application-level messages are destined for an IP-based application server, SUA resolves the destination to an SCTP association or IP address using, for example, techniques such as Global Title Translation. In this fashion, SUA can be used to carry a protocol that uses the transport services of SCCP, but takes place between IP-based nodes.

6.3 SIP: Much More Than Voice

The Session Initiation Protocol (SIP, RFC 3261) is a lightweight IP application protocol which provides call control functionality for IP media sessions. Its capabilities include user location, registration, and basic session signaling. Similar to the Hypertext Transport Protocol (HTTP) of web interactions, SIP leverages the clear-text communication, object structure, addressing, and other aspects of traditional Internet protocols.

A key construct of SIP messages derives from the Multipurpose Internet Mail Extensions (MIME, RFC 2045-2049). Many Internet protocols, including electronic mail and web documents also leverage the MIME structure. For instance, when a web browser (or, "HTTP client") requests a web page from a server, the server creates an object with the familiar "header + body + attachment" structure and returns it via the client/server response. Contained in the "body" of the object are instructions for onscreen formatting of the object, and contained in the "attachments" of the object are MIME-encoded, non-text multimedia components such as images, etc. SIP messages have also adopted this structure. The standard SIP message is composed of a text message in the "header" of a MIME document, with an optional "body" attached. The text message contains a "command" line (a *SIP method*) followed by several other header lines, some mandatory and some optional. Some basic SIP methods include `REGISTER` for registering contact information, `INVITE`, `ACK`, `CANCEL` and `BYE` for setting up and terminating sessions, and `OPTIONS` for querying capabilities. Each header line in the SIP message conveys a certain class of information about the call or the command. A single MIME attachment accompanies SIP messages to describe session parameters. This attachment is a Session Description Protocol (SDP, RFC 2327) "object" which contains parameters regarding the IP port number for particular streams, the compression type and rate, and several other media-specific items. By using the request/response pairs of SIP messages, arbitrary multimedia calls can be negotiated, created, and effectively managed between any number of IP-connected endpoints. The popularity of SIP (partially due to its simplicity and similarity to existing "document object" structures) has given rise to several extensions of the basic protocol for the purpose of handling various situations related to call-state management, endpoint control, and other issues.

An example usage of SIP in which user *A* wants to establish communication with another user *B* is shown in Fig. 15. In the first step, the SIP client for *A* might

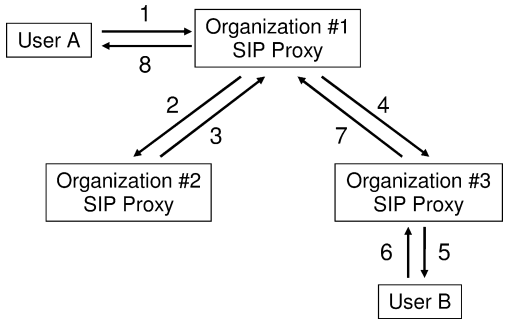


FIG. 15. Example usage of SIP involving user location.

issue an INVITE request to the local SIP proxy of organization-1, where user *A* is located. In the request, user *B* is identified by its SIP URI which might looked like sip:userB@organization2.com. The request would then be forwarded to the SIP proxy of organization-2 (step 2). In the example of Fig. 15, user *B* is located at organization-3 at the time of call. User *B* has also issued periodic REGISTER requests to the SIP proxy at organization-2 to update his present location. In step 3, the present location of user *B* at organization-3 is returned to the SIP proxy of organization-1, which subsequently forwards the INVITE request to the SIP proxy of organization-3 (step 4). The SIP proxy at organization-3 knows where user *B* is located, and forwards the request to the SIP agent of user *B* (step 5). In steps 6–8 of Fig. 15, user *B* accepts the call, and a response value of OK is returned. When user *A* obtains a response of OK for its INVITE request, an ACK is issued to confirm to *B* that an OK has been received, and media communication can proceed. Note that even when SIP proxies are employed, as in the example of Fig. 15, they are only involved in the SIP message exchange but not the actual data transport.

Multiple protocols are typically needed in addition to SIP for realizing an actual call. In particular, SDP is often employed to specify the media types and parameters for the communication, and can be part of the INVITE message. The Real-time Transport Protocol (RTP, RFC 3267), may be used for actual media transport after the initial call setup exchanges. In addition to setup and teardown of a session, some mid-session management is also possible under SIP. Specifically, the INVITE method can be used after communication has started to modify the session, e.g., the number of media streams, media types, and even invite new users to the communication.

6.3.1 Example SIP Usage in 3GPP

There are competing standards, most notably ITU H.323, that address the needs of call initiation. Nevertheless, SIP has gained acceptance in the telecommunications environment as 3GPP has specified the use of SIP for its IP Multimedia subsystem. The advantages of SIP in this case include fewer constraints and less implementation time for vendors. A more detailed comparison of H.323 and SIP for the purpose of 3GPP is available in [49], and some example usage of SIP in 3GPP is outlined in [50].

Figure 16 shows an example in which SIP is used to add an additional user to an existing conversation in a 3G network. In Fig. 16, a conversation is already in-progress between a mobile user *MS* and a user *B* in the legacy public phone network (PSTN). The mobile user wants to add an additional regular phone user *C* to the conversation, and so issues a single INVITE request as shown in step 1. The Call-State Control Function (CSCF) forwards the INVITE request to both *B* and *C* in step 2. Since *C* is not already in the conversation, the Media Gateway Controller

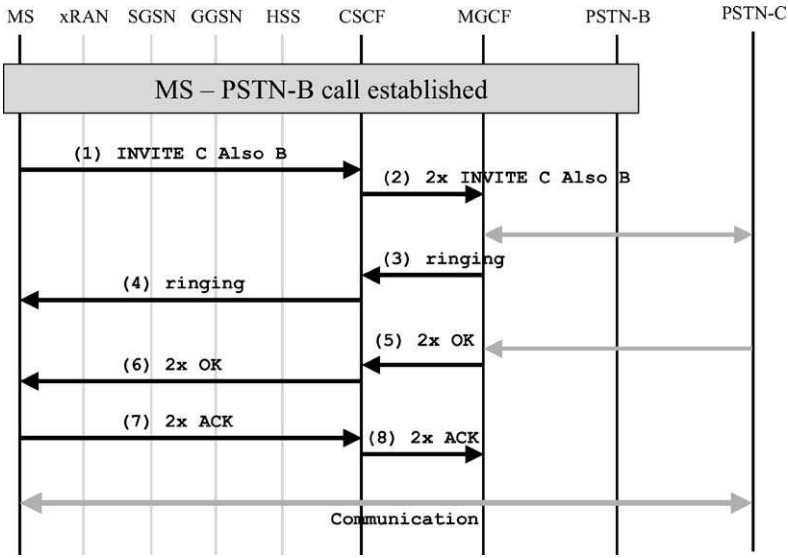


FIG. 16. Example usage of SIP to add a new user (*C*) to an existing conversation between two users *MS* and *B*.

(MGC) issues a series of messages connecting to user *C* while returning a response of `Ringing` to the CSCF (steps 3 and 4). Eventually, the MGCF returns `OK` for both users *B* and *C* for the `INVITE` request (steps 5 and 6). The `OK` responses are forwarded from the CSCF to the mobile user, who can then issue `ACK` requests to signal the completion of the call establishment (steps 7 and 8).

7. Conclusion

As telecommunications and computing technologies converge, the architecture and components of these networks are becoming increasingly commoditized. Fueled in many ways by regulatory issues and the pervasiveness of computing and networking technologies, the convergence phenomenon is inevitable. Whether commoditization takes the form of widespread adoption of standard, open communication interfaces, open-source software, or low-cost industry-standard devices, the message is clear: multi-modal, multi-media communication is a vital part of modern society, and supporting such communications via an IP-based infrastructure requires highly distributed computing technologies. This implicit requirement is a common thread among the technologies that are disrupting conventional telecommunications.

A fundamental premise of converged networking is the migration of enhanced services (customizable behavior, multimedia capabilities, etc.) away from closed, centralized, and provider-owned facilities. This premise pre-supposes a pervasive, common, and easily adapted "intelligent edge," which violates the fundamental assumptions of conventional telephony. The resulting phenomenon is an accelerating spiral. Desire for rapidly evolving services is fueled by the commoditization of network access, bandwidth availability, and ubiquitous computing elements. Development and commoditization of these enabling technologies is driven by demand for ever-more sophisticated services and endless bandwidth. Unfortunately, effective delivery of such services pre-supposes sufficient bandwidth and quality of service guarantees for multimedia and isochronous data streams. Adequate QoS for multimedia streams is difficult for telephony networks, which are structured for voice transport. Adequate QoS for isochronous streams is difficult for IP networks, which are structured for best-effort data. Clearly, the increasing demand for mobile wireless voice *and* Internet access will continue to drive the development of exciting new technologies to satisfy this need, and the spiral of convergence will continue. From signaling to transport protocols, from processors to system architectures, the end-to-end requirements of advanced services are driving toward fully distributed, pervasive computing.

This chapter has highlighted and categorized several technologies, trends, and architectures which are affecting the structure of telecommunications networks. These "disruptive technologies" are clear evidence that the information revolution is affecting all facets of modern communications.

REFERENCES

[1] Christensen C., *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*, Harvard Business School Press, Boston, MA, 1997.

[2] Isenberg D., "The dawn of the stupid network", *ACM Networker Mag.* **2** (1) (1998) 24–31.

[3] "The Internet Engineering Task Force (IETF)". The complete text of IETF Requests for Comments (RFCs) can be retrieved from the IETF web site via the following URL, where 'NNNN' is the RFC number: http://www.ietf.org/rfc/rfcNNNN.txt.

[4] Shaikh F., McClellan S., Singh M., Sannedhi C.K., "End-to-end testing of IP quality of service mechanisms", *IEEE Computer* (2002) 80–87.

[5] Ghanbari M., "Two-layer coding of video signals for VBR networks", *IEEE J. Select. Areas Commun.* **7** (5) (1989) 771–781.

[6] Dagiuklas A., Ghanbari M., "Priority mechanisms in ATM switches carrying two layer video data", *Electronic Letters* **29** (3) (1993) 273–274.

[7] Garrett M., Vetterli M., "Joint source/channel coding of statistically multiplexed real-time services on packet networks", *IEEE/ACM Trans. Networking* **1** (1) (1993) 71–80.

[8] Aravind R., Civanlar M., Reibman A., "Packet loss resilience of MPEG-2 scalable video coding algorithm", *IEEE Trans. Circuits Syst. Video Technol.* **6** (5) (1996) 426–435.
[9] Wilson D., Ghanbari M., "An efficient loss priority scheme for MPEG-2 variable bit rate video for ATM networks", in: *Proc. IEEE Globecom, London, UK*, 1996, pp. 1954–1958.
[10] Leicher C., "Hierarchical encoding of MPEG sequences using priority encoding", Tech. Rep., TR-94-058, ICSI, November 1994, http://www.icsi.berkeley.edu/PET/pet-documents.html.
[11] Heinzelman W., Budagavi M., Talluri R., "Unequal error protection of MPEG-4 compressed video", in: *Proc. Intl. Conf. Image Processing, Kobe, Japan*, vol. 2, 1999, pp. 503–504.
[12] Tan W., Zakhor A., "Data classification schemes for streaming MPEG video over delay and loss differentiated networks", in: *Proc. Packet Video Workshop, Kyongju, Korea*, 2001.
[13] Krunz M., Hughes H., "A performance study of loss priorities for MPEG video traffic", in: *Proc. IEEE Internat. Conf. on Commun., Seattle, WA*, 1995, pp. 1756–1760.
[14] Tan W., Cui W., Apostolopoulos J., "Playback-buffer equalization for streaming media using stateless transport prioritization", in: *Proc. Packet Video Workshop, Nantes, France*, 2003.
[15] Miu A., Apostolopoulos J., Tan W., Trott M., "Low-latency wireless video over 802.11 networks using path diversity", in: *Proc. IEEE Internat. Conf. on Multimedia and Expo (ICME), Baltimore, MD*, 2003.
[16] Matsuoka H., Yoshimura T., Ohya T., "A robust method for soft IP handover", *IEEE Internet Comput.* **7** (2) (2003) 18–24.
[17] Anton B., Bullock B., Short J., "Best current practices for wireless internet service provider (WISP) roaming", Technical Report v. 1.0, Wi-Fi Alliance, February 2003.
[18] Henry P.S., Luo H., "WiFi: What's next?", *IEEE Commun. Mag.* **40** (12) (2002) 66–72.
[19] Mangold S., Choi S., May P., Klein O., Hiertz G., Stibor L., "IEEE 802.11e wireless LAN for quality of service", in: *Proc. Euro. Wireless, Florence, Italy*, 2002, pp. 32–39.
[20] Gu D., Zhang J., "QoS enhancement in IEEE 802.11 wireless local area networks", *IEEE Commun. Mag.* **41** (6) (2003) 120–124.
[21] "IEEE standard for local and metropolitan area networks—port-based network access control", IEEE Std 802.1X-2001 (2001).
[22] Funk P., Blake-Wilson S., "EAP tunneled TLS authentication protocol (EAP-TTLS)", Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-pppext-eap-ttls-03.txt, August 2003.
[23] Palekar A., Simon D., Zorn G., Josefsson S., "Protected EAP protocol (PEAP)", Internet Draft, http://www.ietf.org/internet-drafts/draft-josefsson-pppext-eap-tls-eap-06.txt, March 2003.
[24] Stubblefield A., Ioannidis J., Rubini A., "Using the Fluhrer, Mantin, and Shamir attack to break WEP", Technical Report TD-4ZCPZZ, AT & T Labs, August 2001.
[25] "Advanced Encryption Standard (AES)", Federal Information Processing Standard (FIPS) PUB 197 (2001).
[26] 3GPP, "Transparent end-to-end packet switched streaming service (PSS): General description", TS 26.233 (2001).

[27] Cheung G., Tan W., Yoshimura T., "Double feedback streaming agent for real-time delivery of media over 3G wireless networks", in: *Proc. Wireless Comm. and Networks Conf., New Orleans, LA*, 2003.
[28] Yoshimura T., Ohya T., Kawahara T., Etoh M., "Rate and robustness control with RTP-monitoring agent for mobile wireless streaming", in: *Proc. IEEE Internat. Conf. on Commun., New York, NY*, 2002.
[29] "The Apache project", http://www.apache.org.
[30] "The Mozilla project", http://www.mozilla.org.
[31] "The Linux Kernel project", http://www.kernel.org.
[32] "The GNU General Public License (GPL)", http://www.gnu.org.
[33] "Proprietary kernel modules—the boundary shifts?", http://lwn.net/Articles/13398.
[34] Benton W., "Loadable kernel module exploits", *Linux Journal* **89** (2001) 24–29.
[35] Love R., "Lowering latency in Linux: Introducing a preemptible kernel", *Linux Journal* **97** (2002) 48–52.
[36] Williams C., "Linux scheduler latency", White paper, RedHat Software, March 2002.
[37] Love R., "Introducing the 2.6 kernel", *Linux Journal* **109** (2003) 52–57.
[38] Mosberger D., "A closer look at the Linux O(1) scheduler", http://www.hpl.hp.com/research/linux/kernel/o1.php, 2003.
[39] Drepper U., Molnar I., "The Native POSIX Thread Library for Linux", White paper, Redhat Software, February 2003.
[40] "Carrier grade Linux: Technical scope, requirements, & architecture specifications", Tech. Rep., The Open Source Development Laboratory (OSDL), Beaverton, OR, 2002, http://www.osdl.org/projects/cgl.
[41] "The Linux Standards Base", http://www.linuxbase.org.
[42] "The Service Availability Forum", http://www.saforum.org.
[43] Maloy J., "Telecom Inter-Process Communication", Tech. Rep. LMC/JO-01:006-PA6, Ericsson, January 2003.
[44] Eranian S., Mosberger D., "The Linux/ia64 project: Kernel design and status update", Tech. Rep. HPL-2000-85, Hewlett Packard Internet & Mobile Systems Lab, HP Labs—Palo Alto, CA, June 2000.
[45] Yager T., "AMD Opteron: Building a bridge to 64 bits", *InfoWorld* (2003) 43–48.
[46] Baxter M., "AMD64 Opteron: First look", *Linux Journal* (2003) 58–62.
[47] George T., Bidulock B., Dantu R., Schwarzbauer H.J., Morneault K., "SS7 MTP2-user peer-to-peer adaptation layer", Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-sigtran-m2pa-09.txt, June 2003.
[48] Loughney J., Sidebottom G., Coene L., Verwimp G., Keller J., Bidulock B., "Signalling connection control part user adaptation layer (SUA)", Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-sigtran-sua-15.txt, June 2003.
[49] 3GPP, "A comparison of H. 323v4 and SIP", Tdoc S2-00505 (January 2000).
[50] 3GPP, "Presence service based on session initiation protocol (SIP): Functional models, information flows and protocol details", TR 24.841 v0.5.0, Release 6 (February 2003).

This Page Intentionally Left Blank

Ions, Atoms, and Bits: An Architectural Approach to Quantum Computing

DEAN COPSEY

University of California, Davis
Davis, CA
USA

MARK OSKIN

University of Washington
Seattle, WA
USA
oskin@cs.washington.edu

FREDERIC T. CHONG

University of California, Davis
Davis, CA
USA

Abstract

Computing with quantum states may be the most efficient approach to solving some problems that take exponential time on conventional computers. Quantum states, however, are short-lived without constant error correction. Furthermore, the classical control required by proposed quantum schemes imposes constraints on how a quantum computer can be built.

This chapter focuses on a basic architectural requirement for any computation—communication. Unlike classical signals, quantum data cannot be simply transmitted over a wire, but must be moved step-wise from location to adjacent location. We start with a brief overview of quantum computing operations, error correction, and algorithms. Next we analyze a straightforward communication mechanism—the swapping channel. We compare the swapping channel with a longer-range communication mechanism—the teleportation channel. We finish out the chapter with a discussion of error correction. Error correction is necessary

ISSN: 0065-2458/DOI 10.1016/S0065-2458(03)61005-3
Copyright © 2004 Elsevier Inc.
All rights reserved.

to avoid data corruption. We look in detail at its architectural implications and analyze the associated overhead.

1. Introduction

Many important problems seem to require exponential resources on a classical computer. Quantum computers can solve some of these problems with polynomial resources, which has led a great number of researchers to explore quantum information processing technologies [1–7]. Early-stage quantum computers have involved a small number of components (less than 10) and have utilized molecules in solution and trapped ions [8–11]. To exploit our tremendous historical investment in silicon,

however, solid-state silicon quantum computers are desirable. Promising proposals along these lines have begun to appear [12,13]; these even include ideas which merge atomic physics and silicon micromachining [14]. However, as the number of components grows, quantum computing systems will begin to require the same level of engineering as current computing systems. The process of architectural design used for classical silicon-based systems, of building abstractions and optimizing structures, needs to be applied to quantum technologies.

Even at this early stage, a general architectural study of quantum computation is important. By investigating the potential costs and fundamental challenges of quantum devices, we can help illuminate pitfalls along the way toward a scalable quantum processor. We may also anticipate and specify important subsystems common to all implementations, thus fostering interoperability. Identifying these practical challenges early will help focus the ongoing development of fabrication and device technology. In particular, we find that transporting quantum data is a critical requirement for upcoming silicon-based quantum computing technologies.

Quantum information can be encoded in a number of ways, such as the spin component of basic particles like protons or electrons, or in the polarization of photons. Thus, there are several ways in which we might transfer information. First, we might physically transport particles from one point to another. In a large solid-state system, the logical candidate for information carriers would be electrons, since they are highly mobile. Unfortunately, electrons are also highly interactive with the environment and hence subject to corruption of their quantum state, a process known as *decoherence*. Second, we might consider passing information along a line of quantum devices. This *swapping channel* is, in fact, a viable option for short distances (as discussed in Section 9), but tends to accumulate errors over long distances.

Over longer distances, we need something fundamentally different. We propose to use a technique called *teleportation* [15] and to call the resulting long-distance quantum wire a *teleportation channel* to distinguish from a swapping channel. Teleportation uses an unusual quantum property called *entanglement*, which allows quantum information to be communicated at a distance.[1] To understand the mathematical details and practical implications of teleportation, we will need to cover some background before returning to the subject in Section 3.3.

A striking example of the importance of quantum communication lies in the implementation of error correction circuits. Quantum computation must make use of extremely robust error correction techniques to extend the life of quantum data. We present optimized layouts of quantum error correction circuits based upon quantum bits embedded in silicon, although other technologies share many of the same features and limitations. An overview of quantum technologies my be found in [16,17].

[1] The speed of this channel is, however, limited by the rate at which two classical bits can be transmitted from source to destination, without which the quantum information is ambiguous.

We discover two interesting results from our quantum layouts. First, the recursive nature of quantum error correction results in a H-tree-structured circuit that requires long-distance communication to move quantum data as we approach the root. Second, the reliability of the quantum SWAP operator is perhaps the most important operator for a technology to implement reliably in order to realize a scalable quantum computer.

The remainder of this chapter continues with a brief introduction to quantum computing in Section 2. We describe our assumptions about implementation technologies in Section 4. Next, Section 5 discusses how quantum information can be transported in solid-state technologies. This includes a discussion of short-distance *swapping channels* and the more scalable long-distance *teleportation channels*. Section 2.2 introduces error correction algorithms for quantum systems and discusses the physical layout of such algorithms. Then, Section 6 probes details of two important error correction codes. Following this, in Section 7, we demonstrate the need for teleportation as a long-distance communication mechanism in the layout of recursive error correction algorithms. Finally, Section 8 discusses system bandwidth issues and in Section 9 we conclude.

2. Background

While a bit in a classical computer represents either zero or one, a quantum bit can be thought of as simultaneously representing both states. More precisely, the state of a qubit is described by *probability amplitudes* for measuring states representing zero or one. The amplitudes are complex values, with real and imaginary parts, and only turn into real probabilities upon external observation. Unlike classical probabilistic computation, the amplitudes for different computational pathways can cancel each other out through interference. The actual probabilities are determined by the modulus of the amplitude, which is the amplitude multiplied by its complex conjugate (hereafter referred to, somewhat inaccurately, as the square of the amplitude).

The key to exponential speedup is that quantum computers directly and simultaneously manipulate probability amplitudes to perform a computation. A system with n qubits has the ability to be in 2^n states *simultaneously*, each with its own probability amplitude. For example, two qubits can be in a *superposition* of the states 00, 01, 10, and 11. The work of a quantum computer is to manipulate qubits and the associated amplitude vectors in a useful manner. Any operation on a single qubit can affect all 2^n states. This is often called *quantum parallelism*, and is a useful way to think about what gives quantum computers such high potential speedups over classical computers. However, only one of these 2^n states can ever be measured.

More precisely, measuring a qubit vector is equivalent to calculating the squares of the amplitudes, and probabilistically choosing one state. The amplitude vector then collapses, with a value of one for the chosen state, and zeroes for all other states. For this reason, quantum computers are best at *NP* problems where only a single answer is needed, and the answer can be verified in P-time.

Designers of quantum algorithms must be very clever about how to get useful answers out of their computations. One method is to iteratively skew probability amplitudes in a qubit vector until the probability for the desired value is near 1 and the probability for other values is close to 0. This is used in Grover's algorithm for searching an unordered list of n elements [18] for a key meeting some arbitrary criteria. The algorithm iterates $\sqrt{n}$ times, at which point a qubit vector representing the keys can be measured. The desired key is found with high probability.

Another option is to arrange the computation such that it does not matter which one of many highly probable results is measured from a qubit vector. This method is used in Shor's algorithm for prime factorization of large numbers [19], building upon modular exponentiation of all states and the quantum Fourier transform, an exponentially fast version of the classical discrete Fourier transform. Essentially, the factorization is encoded within the period of a set of highly probable values, from which the desired result can be obtained no matter which value is measured. Since prime factorization of large numbers is the basis of many modern cryptographic security systems, Shor's algorithm has received much attention.

Section 6 discusses Shor's and Grover's algorithms in more detail.

2.1 Basic Quantum Operations

In general, qubits are denoted in Dirac's "bra, ket" notation. $|0\rangle$ represents a qubit in the zero state, and is pronounced "ket zero." A generic qubit, $|\psi\rangle$, is represented by $\alpha|0\rangle + \beta|1\rangle$, where $\|\alpha\|^2$ and $\|\beta\|^2$ are the probabilities of measuring 0 or 1, respectively. $|0\rangle$ and $|1\rangle$ are also sometimes referred to as the computational basis.

Another useful way of thinking about a qubit is the Bloch sphere (see Fig. 1). $|0\rangle$ is up along the $\hat{z}$-axis, and $|1\rangle$ is down. Generically, $|\psi\rangle = \cos\frac{\phi}{2}|0\rangle + \sin\frac{\phi}{2}e^{i\theta}|1\rangle$. Operations on a qubit are equivalent to rotations of the Bloch sphere.[2]

Figure 2 gives a few basic quantum operations that are used in the proposed quantum architecture. These include one-bit operations such as the bit-flip (X), phase-

[2]It is interesting to note that a vector on the Bloch sphere only has two degrees of freedom. This is because all operators are unitary—they preserve a total probability of unity. Hence, the phase for $|0\rangle$ can be divided out, and kept as a constant. All operators are multiplicative, so this global constant makes no difference, and cannot actually be observed. In general, the zero state for any set of qubits can be thought of having a real, non-negative value. Unfortunately, the Bloch sphere model does not scale to multiple qubits, but it is useful as a visualization tool for single-qubit operations on sets of qubits.

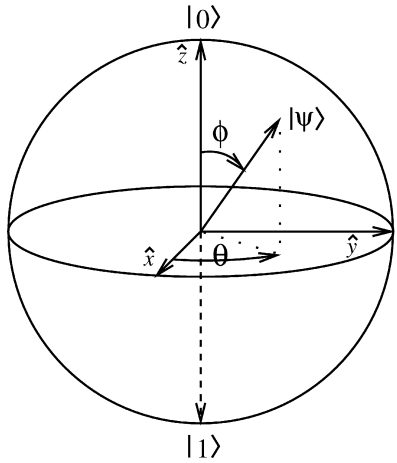


FIG. 1. Bloch sphere representation of a qubit.

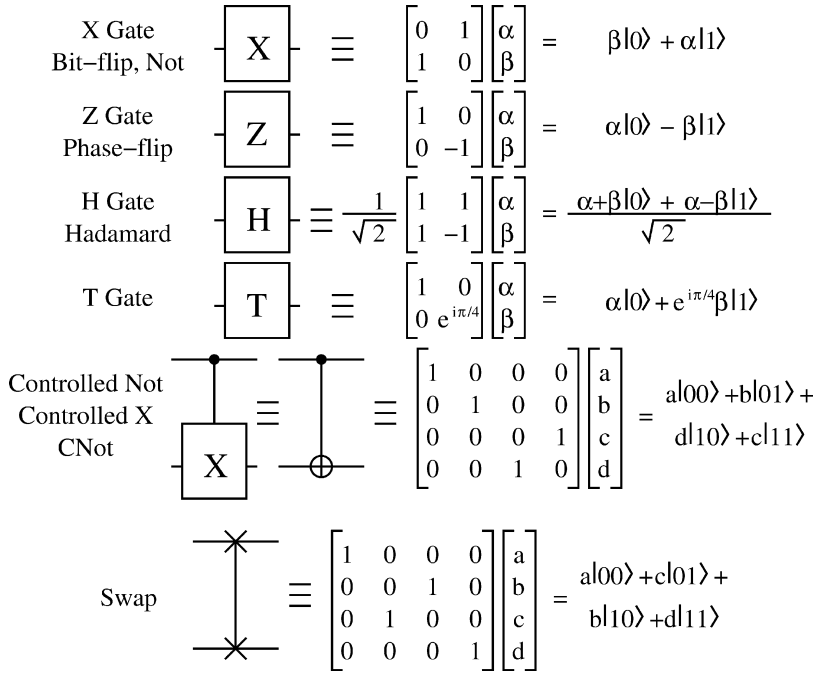


FIG. 2. Basic quantum operations.

flip (Z), Hadamard (H), and "$\pi/8$" (T) gates, as well as the two-bit controlled-not. These are given in both their circuit representation and their matrix representation. The matrix representation involves multiplying the operator by the amplitude vector of the quantum states. The X, Y, and Z operators are equivalent to rotating the Bloch sphere by π around the $\hat{x}$-, $\hat{y}$-, and $\hat{z}$-axes, respectively. The T operator rotates around the $\hat{z}$-axis by $\pi/4$ (it is called $\pi/8$ for historical reasons). Any n-qubit unitary operator may be composed from single-qubit operators and the *CNot* operator. A minimal *universal* set of operators, able to approximate any unitary operator to arbitrary precision, is *CNot*, H, and T.

Another interpretation of the above operators is that the bit-flip exchanges the probabilities of the two states, while the phase flip changes the sign (phase) between them. The Hadamard takes the two states and "mixes" them to a "halfway" state. The controlled-not does a bit-flip if the control qubit is 1: $\text{CNOT}|xy\rangle \to |x, x \oplus y\rangle$, where $\oplus$ is *modulo*-2 addition. These basic gates, along with the measurement of qubits, form the set of operations used for quantum computation.

To illustrate that quantum computation is potentially more powerful than classical computation, it useful to look at *entanglement*. If two qubits are joined to form a system, $|x\rangle|y\rangle \to |xy\rangle$, the result is the *tensor product* (denoted by $\otimes$) of the vector representations:

$$\big(\alpha|0\rangle + \beta|1\rangle\big) \otimes \big(\gamma|0\rangle + \delta|1\rangle\big) \to \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle.$$

Any single-qubit operator, or tensor product of single-qubit operators, may be applied to the system and the qubits remain independent. However, if one applies a Hadamard operator to the first qubit, and then uses that qubit as the control in a CNOT on the second qubit (see Fig. 3), the resulting *superposition* of states cannot be split into the tensor product of two qubits. The two qubits share information that neither qubit has alone, and there is no concept of state for the individual qubits. The qubits are now tied together, or *entangled*: whatever value is measured for the first qubit will also be measured for the second qubit. The amplitudes for $|01\rangle$ and $|10\rangle$ are zero. This particular state is known as an EPR pair (after Einstein, Podolsky and Rosen, who were among the first to investigate such states). It is also called a Bell state, or a "cat" state, after Schrödinger's infamous thought experiment. Cat states are very important, and are used extensively in quantum computation.

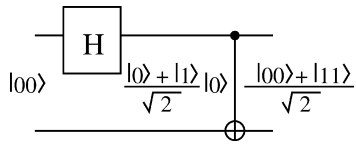


FIG. 3. Creating a "cat" state.

The group of operators given above are sufficient to approximate an arbitrary n-qubit unitary operator to any desired accuracy, although for $n > 1$, the approximation is not necessarily efficient, and may be exponential in n. That said, a few more operators are generally used for descriptions of computations. They are $R_x(\theta)$, $R_y(\theta)$, and $R_z(\theta)$, rotations by θ about the $\hat{x}$-, $\hat{y}$-, and $\hat{z}$-axes. $R_z(\pi/2)$ is used often enough to deserve its own name, S. Any two of the arbitrary-rotation operators can be used to efficiently implement any single-qubit unitary operator, U:

$$U = R_x(\alpha)R_y(\beta)R_x(\gamma)$$

for some α, β, and γ.

Much like classical gates, there are some basic relationships between quantum operators that are important (the † indicates the adjoint, or inverse, of the operator):

$$X^2 = Y^2 = Z^2 = H^2 = I,$$
$$X = X^\dagger,$$
$$Y = Y^\dagger,$$
$$Z = Z^\dagger,$$
$$XZ = \mathrm{i}Y,$$
$$HZH = X,$$
$$HXH = Z,$$
$$S^2 = Z,$$
$$T^2 = S,$$
$$SZ = ZS = S^\dagger,$$
$$SXS^\dagger = Y.$$

There are also several relationships involving CNOT:

(1) X applied to the control input is equivalent to applying X to both outputs.
(2) X applied to the target input is equivalent to applying X to the target output.
(3) Z applied to the control input is equivalent to applying Z to the control output.
(4) Z applied to the target input is equivalent to applying Z to both outputs.
(5) H's applied before and after to the target bit converts a CNOT to a controlled-Z operator.
(6) Two qubits may be swapped with three CNOT's, with the middle CNOT applied in the opposite direction (swapping target and control).

Measurement is the one operation allowed in quantum computation that has no inverse. As stated above, measurement is equivalent to randomly choosing one of the states represented by the qubit(s) based on the probabilities determined by the amplitude vector. Note that measurement destroys the wave function representing the superposition of states. If a qubit is measured as zero, it will be measured as zero from then on, unless another operator is applied, or decoherence occurs. If only part of a set of entangled qubits is measured, the rest of the qubits will be in a superposition of states consistent with the values measured. In the case of the cat state above, if the first qubit is measured as a one, the second qubit will be in a pure $|1\rangle$ state.

Measurements are possible in something other than the computational basis. In terms of the Bloch sphere, the measurement operator described really measures zero for "up" and 1 for "down," but could just as easily measure "left" and "right" (along the $\hat{y}$-axis). Usually, though, such a measurement is made by rotating the $\hat{y}$-axis to the $\hat{z}$-axis ($R_x(\pi/2)$), and then using the usual measurement operator.

A useful application of measurements in other bases and partial measurement is quantum error correction.

2.2 Quantum Error Correction

Quantum phenomena are constantly evolving with time. Atoms decay. Electrons change orbitals by absorbing or emitting photons. Magnetic spin states of nuclei and electrons flip due to external magnetic fields. A quantum system cannot be isolated to the point where it is completely stable. Hence, if two qubits are in an entangled state, they will eventually decohere due to entanglements with the environment. In particular, the environment acts on a qubit every time an operator is applied. The applied operator is a finite approximation implemented by a classical physically controlled process,[3] so the operator itself has a finite chance of introducing an error.

One way to reduce the effect of decoherence is to encode the state of a single logical qubit over several physical qubits. Peter Shor [20] gave the following example:

Imagine that a logical qubit is encoded as $|0_L\rangle = |000\rangle$ and $|1_L\rangle = |111\rangle$. One can measure the difference in value between any two qubits using a circuit like the one in Fig. 4. By performing two such measurements (see Fig. 5) one can determine if a single qubit's value is different than the other two, and correct it (see Table I). When the logical qubit is measured, if one of the qubits is different than the other two, one

[3] Quantum processes, such as an electron changing states, are inherently probabilistic. This is the reason for the tension between the classical and quantum domains. The classical scale components must have enough states to approximate a continuum to a fine enough degree so that errors in the applied operators are kept below an acceptable threshold.

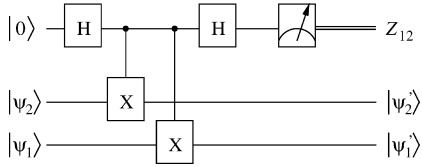


FIG. 4. Measuring Z_{12}, the phase difference between ψ_2 and ψ_1.

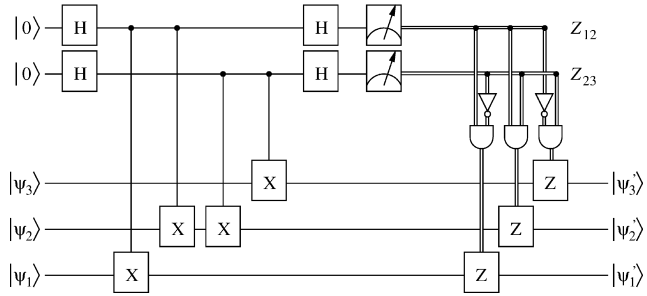


FIG. 5. Syndrome Measurement for 3-bit Code. The meter boxes indicate measurement, and the double lines indicate classical communication controlling the application of the Z operator.

TABLE I
PHASE CORRECTION FOR A 3-QUBIT CODE

Z_{01}	Z_{12}	Error type	Action
0	0	no error	no action
0	1	qubit 3 flipped	flip qubit 3
1	0	qubit 1 flipped	flip qubit 1
1	1	qubit 2 flipped	flip qubit 2

can assume that it was inadvertently flipped along the way. However, it would be better to determine that a qubit had been flipped without having to measure it, since measurement destroys the quantum state. Shor noted that a similar circuit (without the Hadamard gates, and turning the CNOT's around) could be used to measure the difference in phase between two qubits. By encoding each of the three qubits in the phase-flip code with the three-qubit amplitude-flip code (nine qubits total), one could measure and correct any single phase or amplitude error. Furthermore, the process of interacting the extra $|0\rangle$ *ancilla* qubits with the encoded qubits produces an entangled state. After the measurement, the remaining qubits are in a state consistent with the

measurement. That is, if qubit 2 is out of phase with the others, then applying a Z gate will exactly fix the error!

The nine-qubit logical codewords for the states $|0_L\rangle$ and $|1_L\rangle$ are

$$|0_L\rangle = \frac{(|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle) \otimes (|000\rangle + |111\rangle)}{2\sqrt{2}},$$

$$|1_L\rangle = \frac{(|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle) \otimes (|000\rangle - |111\rangle)}{2\sqrt{2}}.$$

The amazing thing about the code is that applying a Z (phase change) operator to each of the nine qubits takes $|0_L\rangle$ to $|1_L\rangle$ and vice versa. It is the same as applying a $\overline{X}$ (logical X) operator to the encoded qubit! Similarly, applying a X operator to each of the physical qubits performs a $\overline{Z}$ operation.

Shor's code, based on the classical error correction method of repetition, is termed a $[\![9, 1, 3]\!]$ code—nine physical qubits, encoding one logical qubit, with a Hamming distance of three. A code with a distance d is able to correct $(d - 1)/2$ errors. The $[\![9, 1, 3]\!]$ code contains all of the elements of the quantum error-correcting codes in this chapter. To avoid actually measuring the state of the individual qubits, ancilla qubits are used. In addition, the measurement of the ancillae determines a unique *error syndrome*, a mapping from the measured values to the operations necessary to correct the error(s). Much like classical linear error codes, measuring the parity of subsets of the bits determines the error syndrome. The parity measurement tells nothing about the absolute value of the bits, just the relative value. Unlike a classical code, however, the parity measurements are made in a variety of bases: the parity measurement in the computational basis tells which bits have amplitude errors, and the parity measurement in the Hadamard-rotated basis (also called the *Bell* basis) tells which bits have phase errors.

Shortly after Shor demonstrated the $[\![9, 1, 3]\!]$ code, he and Calderbank [21], and independently Steane [22], showed how to create quantum error correction codes based on classical linear codes. One important such code is the so-called Steane code (a $[\![7, 1, 3]\!]$ code). Further refinements and generalizations led to *stabilizer* codes, such as the $[\![5, 1, 3]\!]$ code [23], which is the smallest (densest) known encoding of a single qubit; the $[\![8, 3, 3]\!]$ code [24–26], the densest three-qubit code; the $[\![16, 10, 3]\!]$ code and many others. For more on quantum error-correction codes, the reader is directed to the literature [16].

To summarize, errors in quantum circuits are not limited to full phase or bit flips, but can be any complex-valued linear combination of the two. However, when the error syndrome of an error code is determined, the parity measurements collapse

the error waveform in the error-measurement basis. Measuring the error effectively quantizes it so that only X, Y, and Z operators need be applied to correct it.[4]

2.3 Fault-Tolerant Computation

Qubits are subject to decoherence when they interact with the environment. Applying an operator to a qubit is just such an interaction. On the other hand, if an operator could be applied directly to the encoded qubit(s), the qubit could be error-corrected, and any error detected and corrected. Some stabilizer codes allow easy application of some logical operators, as the nine-qubit code demonstrated. The Steane $[\![7, 1, 3]\!]$ code is even more amazing, in that $\overline{X}$ is applied by applying X to all seven encoding qubits. The same is true for the Z, H, Y, and CNOT operators. The S operator requires applying ZS $(= S^{-1})$. The last operator required to create a *universal* set, the T operator, requires a slightly more complicated procedure. Any logical operator may be applied in a fault tolerant manner, as long as the probability of an error for a physical operator, p, is below a certain threshold, $1/c$, where c is the number of ways *two* errors during the operator application and subsequent error correction can cause an erroneous result. For the $[\![7, 1, 3]\!]$ code, c is about 10^4. The overall probability of error for the logical operator is cp^2. That is, at some step in the application of the operator, and subsequent error correction, two errors would have to occur in order for the logical operator to fail.

If a logical qubit is encoded in n physical qubits, it is possible to encode each of those n qubits with an m-qubit code to produce an mn encoding. Such concatenation of codes can reduce the overall probability of error even further. For example, concatenating the $[\![7, 1, 3]\!]$ with itself gives a $[\![49, 1, 7]\!]$ code with an overall probability of error of $c(cp^2)^2$ (see Fig. 6). Concatenating it k times gives $(cp)^{2^k}/c$, while the size of the circuit increases by d^k and the time complexity increases by t^k, where d is the increase in circuit complexity for a single encoding, and t is the increase in operation time for a single encoding. For a circuit of size $p(n)$, to achieve an desired probability of success of $1 - \varepsilon$, then k must be chosen such that [16]:

$$\frac{(cp)^{2^k}}{c} \leqslant \frac{\varepsilon}{p(n)}.$$

The number of gates (operators) to achieve this result is $\mathrm{O}(\mathrm{poly}(\log p(n)/\varepsilon)p(n))$, provided p is below some threshold.

[4]This is not entirely true. The measurement and correction will return a valid codeword, or superposition of codewords. If more than $(d-1)/2$ errors occur, where d is the Hamming distance, then the error syndrome may indicate that no reliable correction is possible. If more than $(d+1)/2$ errors occur, the corrections indicated by the error syndrome may take the code to some erroneous superposition of codewords.

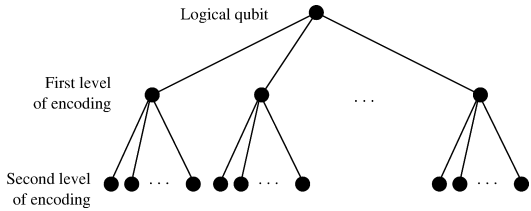


FIG. 6. Tree structure of concatenated codes.

The same results hold for codes other than $[\![7, 1, 3]\!]$, although there is no guarantee that performing logical operations is efficient. For instance, stabilizer codes, allow fairly easy application of the operators X, Y, Z, and H to the encoded qubits. However, for a given stabilizer code, an arbitrary rotation may be difficult to perform in a fault tolerant manner. If an operator requires the application of a two-qubit operator between physical qubits in the same encoding, then an error before or during the application of the operator can propagate to multiple qubits. If the code can only correct a single error, as is the case for all of the distance-three codes discussed, this is already too many.

3. Quantum Algorithms

3.1 Shor's Factoring Algorithm

Perhaps the biggest motivation for research into quantum computation is due to Peter Shor's algorithm for factoring large numbers. Shor's algorithm [19] can factor large numbers in polynomial time on the size of the representation (i.e., the number of bits), using modular exponentiation and an inverse quantum Fourier transform. The best classical algorithms known require exponential time in the number of bits.

Factoring is considered to be a "hard" problem. Rivest, Shamir, and Adleman [27] have used it for the trapdoor function for RSA security.

> "A message is encrypted by representing it as a number M, raising M to a publicly specified power e, and then taking the remainder when the result divided by the publicly specified product, n, of two large secret prime numbers p and q. Decryption is similar; only a different, secret, power d is used, where $e \cdot d = 1 \pmod{((p-1)\cdot(q-1))}$. The security of the system rests in part on the difficulty of factoring the published divisor, n."

Clearly, an algorithm that makes factoring exponentially easier is of immense interest.

A brief outline of Shor's algorithm for factoring n is as follows:

(1) Test the number for the two cases where the algorithm will always fail:
   (a) the number is prime, or
   (b) the number is an integer raised to some power.
   (Both of these tests can be performed in polynomial time.)
(2) Prepare a quantum register of $\lceil \log_2 n \rceil + \varepsilon$ qubits, $|\psi\rangle$, in a superposition of all $2^{\log_2 + \varepsilon}$ states. ε determines the precision of s/r below.
(3) Apply the unitary transform, $U(x)$:

$$U(x)|\psi\rangle|0\ldots01\rangle \to |\psi\rangle\left|x^{\psi} \pmod n\right\rangle$$

for a random x in the interval $[2, n-1]$. The second register is the same size as $|\psi\rangle$, but is prepared as a binary 1. Note that the transform is unitary, since it has an inverse.
(4) Apply the inverse quantum Fourier transform.
(5) Measure $|\psi\rangle$, to get an approximation of s/r, where r is periodicity of the modular exponentiation, i.e., $x^r = 1 \pmod n$, and $1 \leqslant s < r$.
(6) Use Euler's continued fraction algorithm to determine r. The continued fraction algorithm represents a rational number as

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \cdots}},$$

where the a_i are integers. It is the unique, reduced representation for that rational number. If s/r has enough bits, then r is determined uniquely.

$x^r - 1 = kn$, for some integer k. With high probability, one of $x^{r/2} + 1$ or $x^{r/2} - 1$ has a common factor with n, which can be found by using Euler's algorithm to find the greatest common denominator of two integers. Possible reasons for the algorithm to fail are $x^{r/2} = \pm 1 \pmod n$ (probability $\mathrm{O}(n^{-1})$ or inverse exponential in the number of bits to represent n), or one of $x^{r/2} \pm 1$ has a common factor with k, also with a probability of $\mathrm{O}(n^{-1})$.

3.2 Grover's Search Algorithm

Grover's search algorithm is used to find a solution set to a NP-decision problem, given a polynomial time operator to recognize the solution. A familiar example is 3-SAT. 3-SAT is NP complete, so the best known algorithm to find a solution is exponential in the number of literals. However, given a proposed solution to a 3-SAT problem, it is simply a matter of verifying that each clause in the expression evaluates to true, which can be done in polynomial time for a given set of literals.

A unitary operator to evaluate any proposed solution to a 3-SAT problem can be built with a polynomial number of gates. The same is true of any boolean logic that

can be performed in polynomial time on a classical computer. Much like the CNOT gate described above, this operator can be made to flip a single bit if the expression evaluates to true. If the set of qubits to be operated on starts out as a superposition of all states, applying the 3-SAT evaluation operator changes the phase of those states that represent a solution, marking them with a negative phase.

The next operation, G, *inverts about the mean*:

$$G\sum_k \alpha_k|k\rangle = \sum_k \bigl[-\alpha_k + 2\langle\alpha\rangle\bigr]|k\rangle.$$

The operator G is $(2|\psi\rangle\langle\psi| - I)$,[5] where $|\psi\rangle$ is the equal superposition of all states. It is equivalent to $H^{\otimes n}(2|0\rangle\langle 0| - I)H^{\otimes n}$ (n is size of the vector). The expression in parentheses is an operator that changes the phase of all states but the zero state.

Mathematically, this is the same as rotating in a plane defined by $|\psi\rangle$ and $|M\rangle$, the vector of all states that are solutions, by the amount

$$\theta = \arcsin\left(\frac{2\sqrt{|M|(|N| - |M|)}}{|N|}\right)$$

where $|M|$ is the size of the solution set and $|N|$ is the size of the solution space, i.e., 2^n. If this sequence is iterated $\sqrt{|N|}$ times, a solution will be measured with probability O(1). Unfortunately, this is not P-time solution, as was the case with Shor's algorithm. On the other hand, it may make many problems tractable that otherwise would not have been.

3.3 Teleportation

Although many quantum error correction codes could be used in a quantum computer, converting between them can be problematic. In fact, conversion between codes can randomly propagate errors across qubits and compromise reliability. Fortunately, there is a special way to convert between codes that avoids this problem. This method involves the quantum primitive of teleportation [28]. As it turns out, teleportation is not only a good way to convert between codes, but it is also a good way to transport quantum data between the different parts of the system.

Quantum teleportation is the re-creation of a quantum state at a destination using some classical bits that must be communicated along conventional wires or other media. In order for this to work, we need to precommunicate an EPR pair, $|00\rangle + |11\rangle$. We use the information shared by the pair to achieve teleportation.

[5]The "bra" notation, $\langle\psi|$ is the adjoint (conjunct and transpose) of $|\psi\rangle$, the column vector of the wavefunction amplitudes. The notation $|0\rangle\langle 0|$ is a matrix with a 1 in the upper left corner, and zeroes everywhere else.

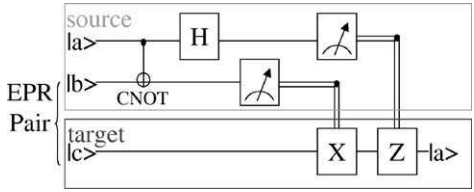


FIG. 7. Quantum teleportation.

Figure 7 gives a more precise view of the process. We start with an EPR pair at the source end of the wire. We separate the pair, keeping one qubit, $|b\rangle$, at the source and transporting the other, $|c\rangle$, to the destination. Note that quantum data is denoted by the single lines, while classical data is represented by double lines. When we want to send a quantum bit of data, $|a\rangle$, we first interact $|a\rangle$ with $|b\rangle$ using a CNOT and a Hadamard gate. We then measure the $|a\rangle$ and $|b\rangle$, and send the two one-bit results to the destination classically, where the receiver uses them to re-create the correct phase and amplitude in $|c\rangle$ such that it takes on the state of $|a\rangle$. The re-creation of phase and amplitude is done with classically controlled-X and Z gates, which perform the same function as the gates described in Fig. 2 but contingent on classical control bits, the measurements of $|a\rangle$ and $|b\rangle$.

Note that the original state of $|a\rangle$ is destroyed once we take our two measurements.[6] Intuitively, since $|c\rangle$ has a special relationship with $|b\rangle$, interacting $|a\rangle$ with $|b\rangle$ makes $|c\rangle$ resemble $|a\rangle$, modulo a phase and/or amplitude error. The two measurements allow us to correct these errors and re-create $|a\rangle$ at the destination.

In order to use teleportation to convert between different error coding schemes, the two halves of our EPR pair are encoded into the source and destination codes. The source and destination qubits are then interacted bitwise with the respective EPR halves just as in the basic teleportation algorithm. The result is a "quantum wire" that gives us a means to both transport quantum data and convert between different error correction codes.

4. Solid-State Technologies

With some basics of quantum operations in mind, we turn our attention to the technologies available to implement these operations. Experimentalists have examined several technologies for quantum computation, including trapped ions [29], photons [30], bulk spin NMR [31], Josephson junctions [32,13], SQUIDS [33], electron spin

[6] This is consistent with the *no-cloning* theorem, which states that an arbitrary quantum state cannot be perfectly copied; this is fundamentally because of the unitarity of quantum mechanics.

resonance transistors [34], and phosphorus nuclei in silicon (the "Kane" model) [12, 35]. Of these proposals, only the last three build upon a solid-state platform; they are generally expected to provide the scalability required to achieve a truly scalable computational substrate.

For the purposes of this paper, the key feature of these solid-state platforms are:

(1) Quantum bits are laid out in silicon in a 2D fashion, similar to traditional CMOS VLSI.
(2) Quantum interactions are near-neighbor between bits.
(3) Quantum bits can not move physically, but quantum data can be swapped between neighbors.
(4) The control structures necessary to manipulate the bits prevent a dense 2D grid of bits. Instead, we have linear structures of bits which can cross, but there is a minimum distance between such intersections that is on the order of 20 bits for our primary technology model [36]. This restriction is similar to a "design rule" in traditional CMOS VLSI.

These four assumptions apply to several solid-state technologies. For concreteness, we will focus upon an updated version of Kane's phosphorus-in-silicon nuclear-spin proposal [35]. This scheme will serve as an example for the remainder of the paper, although we will generalize our results when appropriate.

Figure 8 illustrates important dimensions of the Kane scheme. Shown are two phosphorus atoms spaced 15–100 nm apart. Quantum states are stored in relatively stable electron-donor (e^-–$^{31}P^+$) spin pairs, where the electron (e) and the donor nucleus (n) have opposite spins. The basis states, $|0\rangle$ and $|1\rangle$ are defined as the superposition states $|0\rangle \equiv |\uparrow_e \downarrow_n\rangle + |\downarrow_e \uparrow_n\rangle$ and $|1\rangle \equiv |\uparrow_e \downarrow_n\rangle - |\downarrow_e \uparrow_n\rangle$. Twenty nanometers

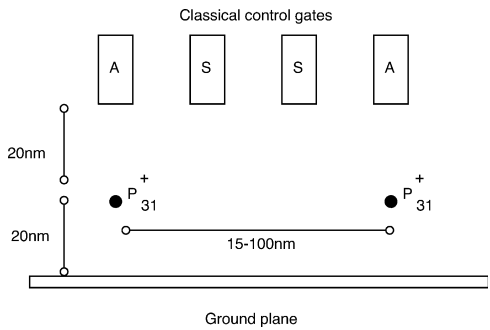


FIG. 8. The basic quantum bit technology proposed by Kane [35]. Qubits are embodied by the nuclear spin of a phosphorus atom coupled with an electron embedded in silicon under high magnetic field at low temperature.

above the phosphorus atoms lie three classical control wires, one A gate and two S gates. Precisely timed pulses on these gates provide arbitrary one- and two-qubit quantum gates.

Single qubit operators are composed of pulses on the A-gates, modulating the hyperfine interaction between electron and nucleus to provide Z axis rotations. A globally applied static magnetic field provides rotations around the X axis. By changing the pulse widths, any desired rotational operator may be applied, including the identity operator. Two-qubit interactions are mediated by S-gates, which move an electron from one nucleus to the next. The exact details of the pulses and quantum mechanics of this technique are beyond the scope of this paper and are described in [35].

Particularly apropos to the next few sections of this paper, however, is the inter-qubit spacing of 15–100 nm. The exact spacing is currently a topic of debate within the physics community, with conservative estimates of 15 nm, and more aggressive estimations of 100 nm. The tradeoff is between noise immunity and difficulty of manufacturing. For our study, we will use a figure (60 nm) that lies between these two. This choice implies that the A and S gates are spaced 20 nm apart. We parameterize our work, however, to generalize for changes in the underlying technology.

The Kane proposal, like all quantum computing proposals, uses classical signals to control the timing and sequence of operations. All known quantum algorithms, including basic error correction for quantum data, require the determinism and reliability of classical control. Without efficient classical control, fundamental results demonstrating the feasibility of quantum computation do not apply (such as the Threshold Theorem used in Section 5.2.3).

Quantum computing systems display a characteristic tension between computation and communication. Fundamentally, technologies that transport data well do so because they are resistant to interaction with the environment or other quantum bits; on the other hand technologies that compute well do so precisely because they *do* interact. Thus, computation and communication are somewhat at odds.

In particular, atomic-based solid-state technologies are good at providing scalable computation but complicate communication, because their information carriers have nonzero mass. The Kane proposal, for example, represents a quantum bit with the nuclear spin of a phosphorus atom implanted in silicon. The phosphorus atom does not move, hence transporting this state to another part of the chip is laborious and requires carefully controlled swapping of the states of neighboring atoms. In contrast, photon-based proposals that use polarization to represent quantum states can easily transport data over long distances through fiber. It is very difficult, however, to get photons to interact and achieve any useful computation. Furthermore, transferring quantum states between atomic and photon-based technologies is currently extremely difficult.

Optimizing these tensions, between communication and computation, between classical control and quantum effects, implies a structure to quantum systems. In this paper, we begin to examine this optimization by focusing on communication in solid-state quantum systems. Specifically, we begin by examining the quantum equivalent of short and long "wires."

5. Transporting Quantum Information: Wires

In this section, we explore the difficulty of transporting quantum information within a silicon substrate. Any optimistic view of the future of quantum computing includes enough interacting devices to introduce a spatial extent to the layout of those devices. This spatial dimension, in turn, introduces a need for wires. One of the most important distinctions between quantum and classical wires arises from the *no-cloning* theorem [2]: quantum information cannot be copied but must rather be *transported* from source to destination (see footnote 6).

Section 5.1 begins with a relatively simple means of moving quantum data via swap operations, called a *swapping channel*. Unfortunately, the analysis of Section 5.2 indicates that swapping channels do not scale well, leading to an alternative called a *teleportation channel*. This long-distance technology is introduced in Section 5.3 and analyzed in Section 5.4.

5.1 Short Wires: The Swapping Channel

In solid-state technologies, a line of qubits is one plausible approach to transporting quantum data. Figure 9 provides a schematic of a *swapping channel* in which information is progressively swapped between pairs of qubits in the *quantum datapath*—somewhat like a bubble sort.[7] Swapping channels require active control from classical logic, illustrated by the *classical control* plane of Fig. 9.

As simple as it might appear, a quantum swapping channel presents significant technical challenges. The first hurdle is the placement of the phosphorus atoms themselves. The leading work in this area has involved precise ion implantation through masks, and manipulation of single atoms on the surface of silicon [37]. For applications where only a few trial devices are desired, slowly placing a few hundred thousand phosphorus atoms with a probe device [38] may be possible. For bulk

[7]For technologies that do not have an intrinsic swap operation, one can be implemented by three controlled-not gates performed in succession. This is a widely known result in the quantum computing field and we refer the interested reader to [2].

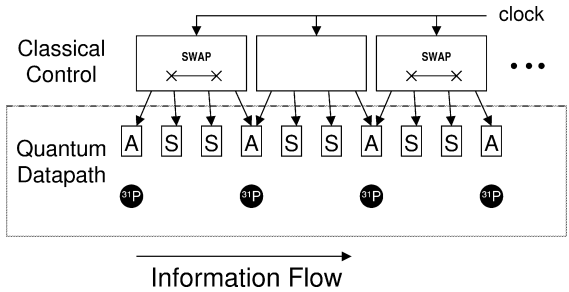


FIG. 9. Short wires are constructed from successive qubits (phosphorus atoms). Information in the quantum data path is swapped from *qubit to qubit under classical control. A single* SWAP *operator requires multiple A- and S-gate voltage pulses. The control circuitry is not to scale.*

manufacturing the advancement of DNA-based or other chemical self-assembly techniques [39] may need to be developed. Note, while new technologies may be developed to enable precise placement, the key for our work is only the spacing (60 nm) of the phosphorus atoms themselves, and the number of control lines (3) per qubit. The relative scale of quantum interaction and the classical control of these interactions is what will lead our analysis to the fundamental constraints on quantum computing architectures.

A second challenge is the scale of classical control. Each control line into the quantum datapath is roughly 10 nm in width. While such wires are difficult to fabricate, we expect that either electron beam lithography [40], or phase-shifted masks [41] will make such scales possible.

A remaining challenge is the temperature of the device. In order for the quantum bits to remain stable for a reasonable period of time the device must be cooled to less than one degree Kelvin. The cooling itself is straightforward, but the effect of the cooling on the classical logic is a problem. Two issues arise: first conventional transistors stop working as the electrons become trapped near their dopant atoms, which fail to ionize. Second, the 10 nm classical control lines begin to exhibit quantum-mechanical behavior such as conductance quantization and interference from ballistic transport [42].

Fortunately, many researchers are already working on low-temperature transistors. For instance, single-electron transistors (SET's) [43] are the focus of intense research due to their high density and low power properties. SET's, however, have been problematic for conventional computing because they are sensitive to noise and operate best at low temperatures. For quantum computing, this predilection for low temperatures is exactly what is needed! Tucker and Shen describe this complementary relationship and propose several fabrication methods in [44].

On the other hand, the quantum-mechanical behavior of the control lines presents a subtle challenge that has been mostly ignored to-date. At low temperatures, and in narrow wires, the quantum nature of electrons begins to dominate over normal classical behavior. For example, in 100 nm wide polysilicon wires at 100 millikelvin, electrons propagate ballistically like waves, through only one conductance channel, which has an impedance given by the quantum of resistance, $h/e^2 \approx 25$ kΩ. Impedance mismatches to these and similar metallic wires make it impossible to properly drive the AC current necessary to perform qubit operations, in the absence of space-consuming impedance matching structures such as adiabatic tapers.

Avoiding such limitations mandates a geometric design constraint: narrow wires must be short and locally driven by nearby wide wires. Using 100 nm as a rule of thumb[8] for a minimum metallic wire width sufficient to avoid undesired quantum behavior at these low temperatures, we obtain a control gate structure such as that depicted in Fig. 10. Here, wide wires terminate in 10 nm vias that act as local gates above individual phosphorus atoms.

Producing a line of quantum bits that overcomes all of the above challenges is possible. We illustrate a design in Fig. 11. Note how access lines quickly taper into upper layers of metal and into control areas of a classical scale. These control areas

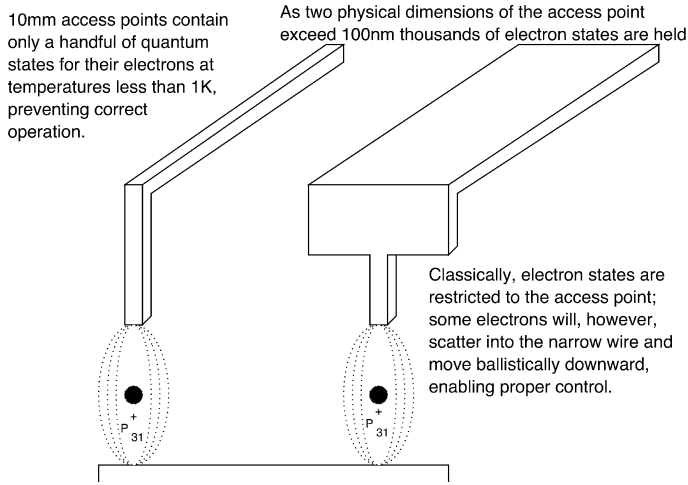


FIG. 10. Quantization of electron states overcome by increasing the physical dimension of the control lines beyond 100 nm. The states propagate quantum-mechanically downward through access vias to control the magnetic field around the phosphorus atoms.

[8]This value is based on typical electron mean free path distances, given known scattering rates and the electron Fermi wavelength in metals.

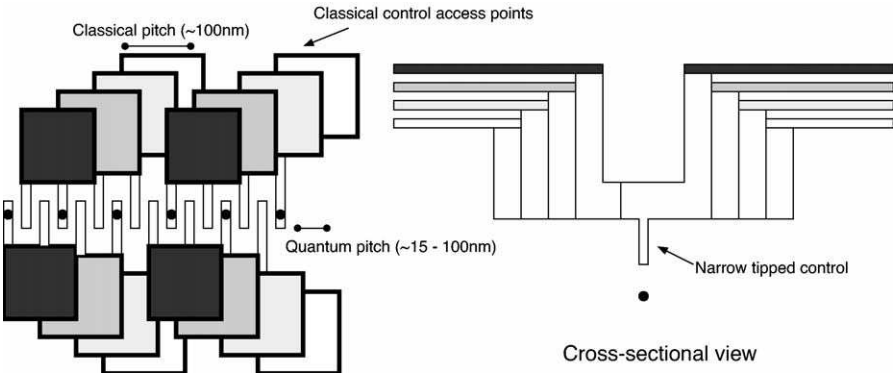


FIG. 11. A linear row of quantum bits: In this figure (not drawn to scale) we depict access control for a line of quantum bits. On the left, we depict a "top down" view. On the right is a vertical cross-section which more clearly depicts the narrow-tipped control lines that quickly expand to classical dimensions.

can then be routed to access transistors that can gate on and off the frequencies (in the 10's to 100's of MHz) required to apply specific quantum gates.

Of course, any solution for data transport must also support routing. Routing is not possible without fanout provided by wire intersections. We can extend our linear row of quantum bits to a four-way intersection capable of supporting sparsely intersecting topologies of quantum bits. We illustrate the quantum intersection in Fig. 12.

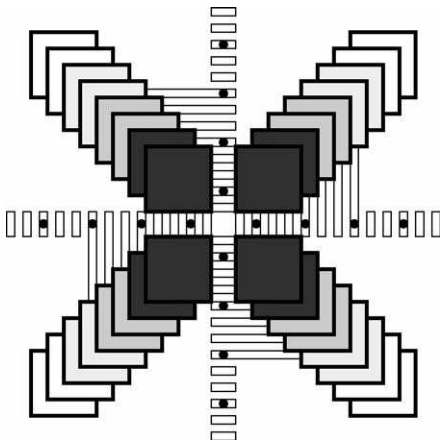


FIG. 12. Intersection of quantum bits. In this simplified view, we depict a four-way intersection of quantum bits. An inversely (diamond shaped) organized junction is also needed to densely pack junction cells.

This configuration is similar to Fig. 11 except that the intersection creates a more challenging tapering.

5.2 Analysis of the Swapping Channel

We now analyze our swapping channel to derive two important architectural constraints: the classical-quantum interface boundary and the latency/bandwidth characteristics. We strive to achieve a loose lower bound on these constraints for a given quantum device technology. While future quantum technologies may have different precise numbers, it is almost certain they will continue to be classically controlled, and thus also obey similar constraints based upon this classical-quantum interface.

5.2.1 Pitch Matching

Our first constraint is derived from the need to have classical control of our quantum operations. As previously discussed, we need a minimum wire width to avoid quantum effects in our classical control lines. Referring back to Fig. 12, we can see that each quadrant of our four-way intersection will need to be some minimum size to accommodate access to our control signals.

Recall from Fig. 8 that each qubit has three associated control signals (one A and two S gates). Each of these control lines must expand from a thin 10 nm tip into a 100 nm access point in an upper metal layer to avoid the effects of charge quantization at low temperatures (Fig. 10). Given this structure, it is possible to analytically derive the minimum width of a line of qubits and its control lines, as well as the size of a four-way intersection. For this minimum size calculation, we assume all classical control lines are routed in parallel, albeit spread across the various metal layers. This parallel nature makes this calculation trivial under normal circumstances (sufficiently "large" lithographic feature size λ_c), with the minimum line segment being equal in length to twice the classical pitching, 150 nm in our case, and the junction size equal to four times the classical pitching, 400 nm, in size. However, we illustrate the detailed computation to make the description of the generalization clearer. We begin with a line of qubits.

Let N be the number of qubits along the line segment. Since there are three gates (an A and two S lines) we need to fit in $3N$ classical access points of 100 nm in dimension each, in the line width. We accomplish this by offsetting the access points in the x and y dimensions (Fig. 11) by 20 nm. The total size of these offsets will be 100 nm divided by the qubit spacing 60 nm times the number of control lines per qubit (3), times the offset distance of 20 nm. This number $100\ \text{nm}/60\ \text{nm} \times 3 \times 20\ \text{nm} = 100\ \text{nm}$ is divided by 2 because the access lines are spread out on each side of the wire. Hence, the minimum line segment will be $100\ \text{nm} + 50\ \text{nm}$. Shorter line segments within larger, more specialized cells are possible.

Turning our attention to an intersection (Fig. 12), let N be the number of qubits along each "spoke" of the junction. We need to fit $3N$ classical access points in a space of $(60\text{ nm} \times N)^2$, where each access point is at least 100 nm on a side. As with the case of a linear row of bits, a 20 nm x and y shift in access point positioning between layers is used for via access. Starting with a single access pad of 100 nm, we must fit 100 nm/60 nm $\times$ 3 additional pads shifted in x and y within the single quadrant of our intersection. This leads to a quadrant size of $100\text{ nm} + 100\text{ nm}/60\text{ nm} \times 3 \times 20\text{ nm} = 200\text{ nm}$. Therefore, the minimum size four way intersection is 8 (rounding up) qubits in each direction.

In this construction we have assumed a densely packed edge to each spoke; however, this is easily "unpacked" with a specialized line segment, or by joining to another junction that is constructed inversely from that shown in Fig. 12. Obviously, the specific sizes will vary according to technological parameters and assumptions about control logic, but this calculation illustrates the approximate effect of what appears to be a fundamental tension between quantum operations and the classical signals that control them. A minimum intersection size implies minimum wire lengths, which imply a minimum size for computation units.

5.2.2 Technology Independent Limits

Thus far we have focused our discussion on a particular quantum device technology. This has been useful to make the calculations concrete. Nevertheless, it is useful to generalize these calculations to future quantum device technologies. Therefore we parameterize our discussion based on a few device characteristics:

Assuming two-dimensional devices (i.e., not a cube of quantum bits), let p_c be the classical pitching required, and p_q the quantum one. Furthermore, let R be the ratio p_c/p_q of the classical to quantum distance for the device technology, m be the number of classical control lines required per quantum bit, and finally λ_c be the feature size of the lithographic technology. We use two separate variables p_c and λ_c to characterize the "classical" technology because they arise from different physical constraints. The parameter λ_c comes from the lithographic feature size, while p_c (which is a function of λ_c) is related to the charge quantization effect of electrons in gold. With the Kane technology we assume a spacing p_q of 60 nm between qubits, three control lines per bit of 100 nm (p_c) each, and a λ_c of 5 nm. We can use these to generalize our pitch matching equations. Here we find that the minimum line segment is simply equivalent to $R(1 + 2\lambda_c m/p_q)$ qubits in length.

Examining our junction structure (Fig. 12), we note that it is simply four line segments, similar to those calculated above, except that the control lines must be on the same side. Therefore the minimum crossing size of quantum bits in a two-dimensional device is of size $\approx 2R(1 + 4\lambda_c m/p_q)$ on a side.

5.2.3 Latency and Bandwidth

Calculating the latency and bandwidth of quantum wires is similar to but slightly different than it is for classical systems. The primary difficulty is decoherence—i.e., quantum noise. Unlike classical systems, if you want to perform a quantum computation, you cannot simply re-send quantum information when an error is detected. The no-cloning theorem prohibits transmission by duplication, thereby making it impossible to re-transmit quantum information if it is corrupted. Once the information is destroyed by the noisy channel, you have to start the entire computation over ("no-cloning" also implies no checkpointing of intermediate states in a computation). To avoid this loss, qubits are encoded in a sufficiently strong error-correcting code that, with high probability, will remain coherent for the entire length of the quantum algorithm. Unfortunately, quantum systems will likely be so error-prone that they will probably execute right at the limits of their error tolerance [45].

Our goal is to provide a quantum communication layer which sits below higher level error correction schemes. Later, in Section 8, we discuss the interaction of this layer with quantum error correction and algorithms. Consequently, we start our calculation by assuming a channel with no error correction. Then we factor in the effects of decoherence and derive a maximum wire length for our line of qubits.

Recall that data traverses the line of qubits with SWAP gates, each of which takes approximately 1 μs to execute in the Kane technology. Hence, to move quantum information over a space of 60 nm requires 0.57 μs. A single row of quantum bits has latency:

$$t_{\text{latency}} = d_{\text{qubits}} \times 1\ \mu s \tag{1}$$

where d_{qubits} is the distance in qubits, or the physical distance divided by 60 nm. This latency can be quite large. A short 1 μm has a latency of 17 μs. On the plus side, the wire can be fully pipelined and has a sustained bandwidth of $1/1\ \mu s = 1$ Mqbps (one million quantum bits per second). This may seem small compared to a classical wire, but keep in mind that quantum bits can enable algorithms with exponential speedup over the classical case.

The number of error-free qubits is actually lower than this physical bandwidth. Noise, or decoherence, degrades quantum states and makes the true bandwidth of our wire less than the physical quantum bits per second. Bits decohere over time, so longer wires will have a lower bandwidth than shorter ones.

The stability of a quantum bit decreases with time (much like an uncorrected classical bit) as a function e^{-kt}. Usually, a normalized form of this equation is used, $e^{-\lambda t}$, where t in this new equation is the number of operations and λ is related to the time per operation and the original k. As quantum bits traverse the wire they arrive

with a fidelity that varies inversely with latency, namely:

$$\text{fidelity} = e^{-\lambda t_{\text{latency}}}. \tag{2}$$

The true bandwidth is proportional to the fidelity:

$$\text{bandwidth}_{\text{true}} = \text{bandwidth}_{\text{physical}} \times \text{fidelity}. \tag{3}$$

Choosing a reasonable[9] value of $\lambda = 10^{-6}$, we find the true bandwidth of a wire to be:

$$\frac{1}{1\ \mu\text{s}} e^{-10^{-6} \times d_{\text{qubits}}} \tag{4}$$

which for a 1 μm wire is close to the ideal (999,983 qbps).

This does not seem to be a major effect, until you consider an entire quantum algorithm. Data may traverse back and forth across a quantum wire millions of times. It is currently estimated [47] that a degradation of fidelity more than 10^{-4} makes arbitrarily long quantum computation theoretically unsustainable, with the practical limit being far higher [45]. This limit is derived from the Threshold Theorem, which relates the decoherence of a quantum bit to the complexity of correcting this decoherence (as discussed in detail, in Section 2.2) [48,49,47].[10] Given our assumptions about λ, the maximum theoretical wire distance is about 6 μm.

5.2.4 Technology Independent Metrics

Our latency and bandwidth calculations require slightly more device parameters. Let t_{swap} be the time per basic SWAP operation. Some technologies will have an intrinsic SWAP, and others will require synthesizing the SWAP from 3 CNOT operations. Let λ be the decoherence rate, which for small λ and t_{swap} is equivalent to the decoherence a quantum bit undergoes in a unit of operation time t_{swap}. This makes the latency of a swapping channel wire equal to:

$$t_{\text{latency}} = d_{\text{qubits}} t_{\text{swap}} \tag{5}$$

where the distance d_{qubits} is expressed in the number of qubits. The bandwidth is proportional to the fidelity or:

$$\text{bandwidth}_{\text{true}} = \frac{1}{t_{\text{swap}}} e^{-\lambda d_{\text{qubits}}}. \tag{6}$$

[9]This value for λ is calculated from a decoherence rate of 10^{-6} per operation, where each operation requires 1 μs. It is aggressive, but potentially achievable with phosphorus atoms in silicon [46,35].

[10]By "practical" we mean without an undue amount of error correction. The threshold theorem ensures that theoretically we can compute arbitrarily long quantum computations, but the practical overhead of error correction makes the real limit 2–3 orders of magnitude higher [45].

This bandwidth calculation is correct so long as the fidelity remains above the critical threshold $C \approx 10^{-4}$ required for fault tolerant computation. Finally, the maximum distance of this swapping channel is the distance when the fidelity drops below the critical threshold:

$$d_{\text{qubits,max}} = \frac{\ln(1-C)}{-\lambda}. \tag{7}$$

No amount of error correction will be robust enough to support a longer wire, while still supporting arbitrarily long quantum computation. For this we need a more advanced architecture. One obvious option is to break the wire into segments and insert "repeaters" in the middle. These quantum repeaters are effectively performing state restoration (error correction). However, we can do better, which is the subject of the next section.

5.3 Long Wires: The Teleportation Channel

In this section, we introduce an architecture for quantum communication over longer distances in solid-state technologies, shown in Fig. 13. This architecture makes use of the quantum primitive of teleportation (described earlier in Section 3.3). In the next few sections, we provide a brief introduction to the core components of this architecture.

Although teleportation and the mechanisms described in this section are known in the literature, what has been missing is the identification and analysis of which mechanisms form fundamental building blocks of a realistic system. In this section, we highlight three important architectural building blocks: the *entropy exchange unit*, the

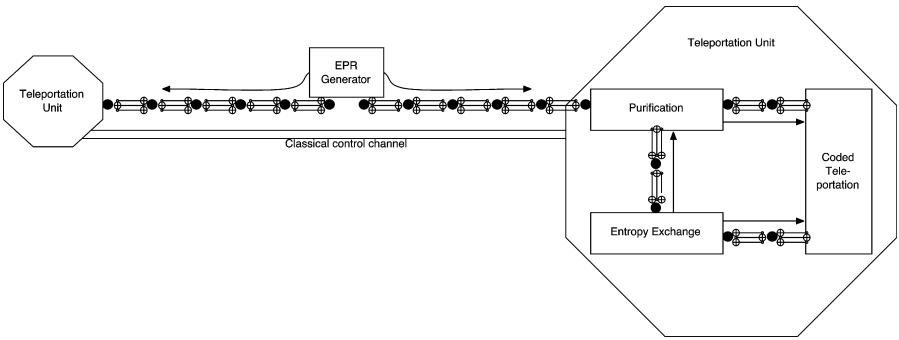


FIG. 13. Architecture for a Quantum Wire: Solid double lines represent classical communication channels, while chained links represented a quantum swapping channel. Single lines depict the direction in which the swapping channel is being used for transport.

EPR generator, and the *purification unit*. Note that the description of theses blocks is quasi-classical in that it involves input and output ports. Keep in mind, however, that all operations (except measurement) are inherently reversible, and the specification of input and output ports merely provides a convention for understanding the forward direction of computation.

5.3.1 Entropy Exchange Unit

The physics of quantum computation requires that operations are reversible and conserve energy. The initial state of the system, however, must be created somehow. We need to be able to create $|0\rangle$ states. Furthermore, decoherence causes qubits to become randomized—the entropy of the system increases through qubits coupling with the external environment.

Where do these zero states come from? The process can be viewed as one of thermodynamic cooling. "Cool" qubits are distributed throughout the processor, analogous to a ground plane in a conventional CMOS chip. The "cool" qubits are in a nearly zero state. They are created by measuring the qubit, and inverting if $|1\rangle$. The measurement process itself requires a source of cold spin-polarized electrons (created, for example, using a standard technique known as optical pumping [46,50]).

As with all quantum processes, the measurement operation is subject to failure, but with high probability leaves the measured qubit in a known state from which $|0\rangle$'s may be obtained. To arbitrarily increase this probability (and make an extremely cold zero state) we can use a technique called *purification*. Specifically, one realization employs an efficient algorithm for data compression [51,52] that gathers entropy across a number of qubits into a small subset of high-entropy qubits. As a result, the remaining qubits are reinitialized to the desired pure, $|0\rangle$ state.

5.3.2 EPR Generator

Constructing an EPR pair of qubits is straightforward. We start with two $|0\rangle$ state qubits from our entropy exchange unit. A Hadamard gate is applied to the first of these qubits. We then take this transformed qubit that is in an equal superposition of a zero and a one state and use it as the control qubit for a CNOT gate. The target qubit that is to be inverted is the other fresh $|0\rangle$ qubit from the entropy exchange unit. A CNOT gate is a qubit like a classical XOR gate in that the target qubit is inverted if the control qubit is in the $|1\rangle$ state. Using a control qubit of $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and a target qubit of $|0\rangle$ we end up with a two-qubit entangled state of $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$: an EPR pair.

The overall process of EPR generation is depicted in Fig. 14. Schematically the EPR generator has a single quantum input and two quantum outputs. The input is

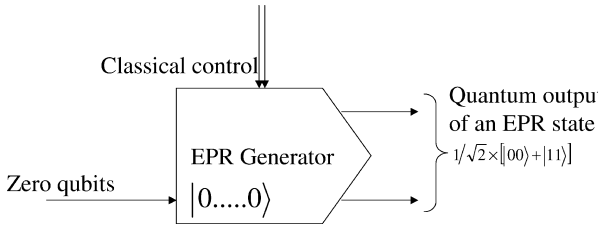


FIG. 14. Quantum EPR generator: Solid double lines represent classical communication (or control), while single lines depict quantum wires.

directly piped from the entropy exchange unit and the output is the entangled EPR pair.

5.3.3 EPR Purification Unit

The final building block we require is the EPR purification unit. This unit takes as input n EPR pairs which have been partially corrupted by errors, and outputs nE asymptotically perfect EPR pairs. E is the entropy of entanglement, a measure of the number of quantum errors which the pairs suffered. The details of this entanglement purification procedure are beyond the scope of this paper but the interested reader can see [53–55].

Figure 15 depicts a purification block. The quantum inputs to this block are the input EPR states and a supply of $|0\rangle$ qubits. The outputs are pure EPR states. Note that the block is carefully designed to correct only up to a certain number of errors; if more errors than this threshold occur, then the unit fails with increasing probability.

Figure 13 illustrates how we use these basic building blocks and protocols for constructing our teleportation channel. The EPR generator is placed in the middle of the wire and "pumps" entangled qubits to each end (via a pipelined swapping channel). These qubits are then purified such that only the error-free qubits remain. Purifi-

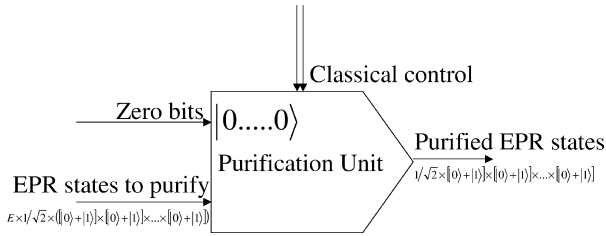


FIG. 15. Quantum purification unit: EPR states are sufficiently regular that they can be purified at the ends of a teleportation channel.

cation and teleportation consume zero-state qubits that are supplied by the entropy exchange unit. Finally, the coded-teleportation unit transmits quantum data from one end of the wire to the other using the protocol described in Section 3.3. Our goal now is to analyze this architecture and derive its bandwidth and latency characteristics.

5.4 Analysis of the Teleportation Channel

The bandwidth of a teleportation channel is proportional to the speed with which reliable EPR pairs are communicated. Since we are communicating unreliable pairs we must purify them, so the efficiency of the purification process must be taken into account. Purification has an efficiency roughly proportional to the fidelity of the incoming, unpurified qubits [51]:

$$\text{purification}_{\text{efficiency}} \approx \text{fidelity}^2. \tag{8}$$

Entropy exchange is a sufficiently parallel process that we assume enough zero qubits can always be supplied. Therefore, the overall bandwidth of this long quantum wire is:

$$1/1\ \mu\text{s} \times \text{e}^{-2\times 10^{-6}\times d_{\text{qubits}}} \tag{9}$$

which for a 1 μm wire is 999,967 qbps. Note this result is less than for the simple wiring scheme, but the decoherence introduced on the logical qubits is only $O(e^{-\lambda\times 10})$. It is this latter number that does not change with wire length which makes an important difference. In the previous short-wire scheme we could not make a wire longer than 6 μm. Here we can make a wire of arbitrary length. For example a wire that is 10 mm long has a bandwidth of 716,531 qbps, while a simple wire has an effective bandwidth of zero at this length (for computational purposes).

The situation is even better when we consider latency. Unlike the simple wire, the wire architecture we propose allows for the pre-communication of EPR pairs at the sustainable bandwidth of the wire. These pre-communicated EPR pairs can then be used for transmission with a constant latency. This latency is roughly the time it takes to perform teleportation, or ≈ 20 μs. Note this latency is much improved compared to the distance-dependent simple wiring scheme.

Using the same constants defined above for the swapping channel, we can generalize our analysis of teleportation channels. The latency is simply:

$$t_{\text{latency}} \approx 10\, t_{\text{swap}}. \tag{10}$$

The bandwidth is:

$$\text{bandwidth}_{\text{true}} = \frac{1}{t_{\text{swap}}}\text{e}^{-2\lambda d_{\text{qubits}}}. \tag{11}$$

Unlike the short wire, this bandwidth is *not* constrained by a maximum distance related to the Threshold Theorem since teleportation is unaffected by distance. The communication of EPR pairs before teleportation, however, can be affected by distance, but at a very slow rate. While purification must discard more corrupted EPR pairs as distance increases, this effect is orders-of-magnitude smaller than direct data transmission over short wires and is not a factor in a practical silicon chip of up to 10's of millimeters on a side.

5.5 Layout of Error-Correction Circuits

While our high-level analysis shows that recursive error correction has desirable efficiency properties, we shall see that the details of implementing such schemes will reveal some key issues. The most important of these issues is the need for reliable, long-distance communication.

Given the pitch-matching constraints of linearity with infrequent junctions from Section 5.2.1, there are still several ways to lay out physical and logical qubits. Optimally, qubits should be arranged to minimize communication overhead.

In a fault tolerant design, the main activity of a quantum computer is error correction. To minimize communication costs, qubits in an encoding block should be in close proximity. Assuming that the distance between junctions is greater than the number of qubits in the block, the closest the qubits can be is in a straight line.

A concatenated code requires a slightly different layout. Error correction is still the important operation, but the logical qubits at all but the bottom level of the code are more complicated. For the second level, the qubits are themselves simple encodings, and so can be laid out linearly. However, we want these qubits in as close proximity to each other as possible, for the same reasons we wanted the qubits in the simple code close. Hence, we need to arrange the bottom level as branches coming off of a main bus. Similarly, the third level would have second-level branches coming off of a main trunk, and so on for higher levels.

In the next two sections, we describe a basic error correction algorithm and its recursive application, focusing on illustrating realistic space and time costs such as those described above, imposed by two-dimensional implementation technologies.

6. Error Correction Algorithms

6.1 The $[\![7, 1, 3]\!]$ Code

Error correcting using the $[\![7, 1, 3]\!]$ code consists of measuring the error syndrome parities of the encoding qubits in various bases, and correcting the codeword based

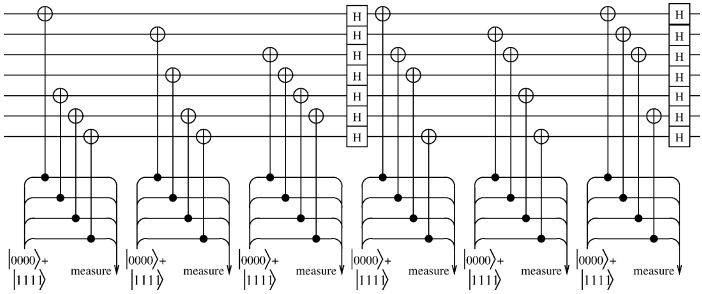


FIG. 16. Measuring the error syndrome for the $[\![7, 1, 3]\!]$ error-correction code.

on the measured syndrome. As shown in Fig. 16, the qubits are rotated to the different measurement bases with Hadamard gates. Parity is then measured in much the same way as with a classical code, using two-qubit CNOT operators acting as XOR's. Conceptually, the parity can be measured in the same way as the three-qubit code in Section 2.2, gathering the parity on ancilla $|0\rangle$'s. To perform a fault tolerant measurement, however, a cat state is used in place of a $|0\rangle$. Figure 16 shows all six parity measurements using cat states. Not shown are cat-state creation and cat-state verification.

A parity measurement consists of the following:

(1) Prepare a cat state from four ancillae, using a Hadamard gate and three CNOT gates.
(2) Verify the cat state by taking the parity of each pair of qubits. If any pair has odd parity, return to step 1. This requires six additional ancillae, one for each pair.
(3) Perform a CNOT between each of the qubits in the cat state and the data qubits whose parity is to be measured (See Fig. 16).
(4) Uncreate the cat state by applying the same operators used to create it in reverse order. After applying the Hadamard gate to the final qubit, $|A_0\rangle$, that qubit contains the parity.
(5) Measure $|A_0\rangle$:
    (A) With $|A_0\rangle = \alpha|0\rangle + \beta|1\rangle$, create the three-qubit state, $\alpha|000\rangle + \beta|111\rangle$ by using $|A_0\rangle$ as the control for two CNOT gates, and two fresh $|0\rangle$ ancillae as the targets.
    (B) Measure each of the three qubits.
(6) Use the majority measured value as the parity of the cat state.

Each parity measurement has a small probability of introducing an error, either in the measurement, or in the data qubits. Hence, the entire syndrome measurement

must be repeated until two measurements agree. The resulting syndrome determines which, if any, qubit has an error, and which X, Z, or Y operator should be applied to correct the error. After correction, the probability of an error in the encoded data is $O(p^2)$.

For the Steane $[\![7, 1, 3]\!]$ code, each parity measurement requires twelve ancillae—four for the cat state to capture the parity, six to verify the cat state, and two additional qubits to measure the cat state. The six parity measurements are each performed at least twice, for a minimum of 144 ancillae to measure the error syndrome!

The minimum number of operations required for an error correction is 38 Hadamards, 288 CNOT's, and 108 measurements. With parallelization, the time required for the operations is $24S + 156C + M$, where S is the time required for a single qubit operator, C is the time required for a CNOT, and M is the time required for a measurement. (We assume all but the last measurement are performed in parallel with other operations.)

6.2 Concatenated Codes

The $[\![7, 1, 3]\!] \times [\![7, 1, 3]\!]$ two-level concatenated code is measured in the same way as the $[\![7, 1, 3]\!]$ code, except the qubits are encoded, and each parity measurement uses a 12-qubit cat state.[11]

The error syndrome measurement is analogous to the singly-encoded $[\![7, 1, 3]\!]$ case, except that the lower-level encodings must be error corrected between operations:

(1) Prepare 12 ancillae in a cat state.
(2) Verify the cat state (66 ancillae for pairwise verification).
(3) Perform CNOT's between the cat state qubits and the qubits encoding the data qubits whose parity is to be measured.
(4) Error correct the four logical data qubits.
(5) Uncreate the cat state, and measure the resulting qubit.

As in the singly-encoded case, each syndrome measurement must be repeated, in this case at least four times. The resulting syndrome determines which, if any, logical qubit has an error. The appropriate $\overline{X}$, $\overline{Z}$, or $\overline{Y}$ operator can be applied to correct the error. After the correction operator is applied to a logical qubit, that qubit must be error-corrected. The probability of an error in the encoded data is $O(p^4)$ after correction.

[11] In the $[\![7, 1, 3]\!]$ code, an $\overline{X}$ consists of an X on each qubit. The parity of the logical qubit is the same as that of the physical qubits. Since a logical qubit is a valid codeword, a four-qubit subset of the qubits has even parity, and the remaining three qubits has the same parity as the logical qubit.

Each parity measurement requires 154 Hadamards, 1307 CNOT's, and 174 measurements, in time $26S + 201C + M$, using the same assumptions as for the non-concatenated case.

Of course, the $[\![7, 1, 3]\!]$ code can be concatenated more than once. The error-correction procedure for higher levels of concatenation is similar to the above. The key is that probability of error for each parity measurement must be $O(p^{2^k})$, for a code concatenated $k - 1$ times.

7. Communication Costs and Error Correction

In this section, we model the communication costs of the error correction algorithms of Section 6, under the constraint of having only near neighbor interactions. While it has previously been proven that under such constraints, the Threshold Theorem can still be made to apply (given suitably reduced failure probability thresholds) [56], a detailed study was not performed with layout constraints on quantum error correction circuits. We first study the growth rate of errors when using SWAP operations. Second, we analyze quantum teleportation as an alternative to SWAP operations for long-distance communication. Finally, we show that teleportation is preferable both in terms of distance and in terms of the accumulating probability of correlated errors between redundant qubits in our codewords.

7.1 Error Correction Costs

The error correction algorithms in the previous section are presented for the ideal situation, where any qubit can interact with any other qubit. Usually, qubits can only interact with their near neighbors, so before applying a two-qubit operator, one of the operand qubits must be moved adjacent to the other.

One of the easiest ways to move quantum data is to use the SWAP operator. By applying SWAP's between alternating pairs of qubits, the values of alternating qubits are propagated in one direction, while the remaining qubit values are propagated in the reverse direction. This swapping channel can be used to supply $|0\rangle$ ancillae for the purpose of error correction, remove "used" ancillae, and allow for qubit movement. Figure 17 illustrates this for the three-qubit example, using two columns of qubits, one for the data and cat-state qubits, and one for communication.

The same layout can be applied to the $[\![7, 1, 3]\!]$ code, giving a minimum time for an error correction parity check of

$$t_{\text{ecc}} = 12(t_{\text{cc}} + t_{\text{cv}} + t_{\text{p}} + t_{\text{cu}} + t_{\text{m}}) \tag{12}$$

where

t_{cc} is the time for cat-state creation;

t_{cv} is the time for cat-state verification;

t_p is the time to entangle the cat state with the parity qubits;

t_{cu} is the time to uncreate the cat state; and

t_m is the time to perform a triply-redundant measurement.

For $[\![7, 1, 3]\!]$ in the ideal, parallel, "sea-of-qubits" model, $t_{cc} = t_{single} + 3t_{cnot}$, $t_{cv} = 6t_{cnot} + t_{meas}$, $t_p = t_{cnot}$, and $t_{cu} = 3t_{cnot} + t_{single}$, where

t_{single} is the time required for a single-qubit operator;

t_{cnot} is the time required for a CNOT operator;

t_{swap} is the time required for a SWAP operator; and

t_{meas} is the time required for redundant measurement.

If communication by swapping is used,

$$t_{cc} = \max(t_{single}, t_{swap}) + 6t_{swap} + 3\max(t_{cnot}, t_{swap}), \tag{13}$$

$$t_{cv} = \max(t_{single}, t_{swap}) + 9t_{swap} + 11\max(t_{cnot}, t_{swap}), \tag{14}$$

$$t_p \leqslant 7t_{swap} + 4\max(t_{cnot}, t_{swap}), \quad \text{and} \tag{15}$$

$$t_{cu} = t_{swap} + 3t_{cnot} + t_{single} + t_{meas}. \tag{16}$$

In the Kane model, $t_{single} < t_{swap} < t_{cnot} < t_{meas}$. Including parallelism between parity measurements, the minimum time for a syndrome measurement is

$$t_{ecc} = 221t_{swap} + 210t_{cnot} + t_{single} + t_{meas}.$$

Since measurement is fully parallelizable, these times assume that there are enough measurement units to perform measurement in parallel with the other operations in the error-correction cycle.

7.2 Multilevel Error Correction

For the singly concatenated code, the data movement in the upper level is more complicated, although Eq. (12) still holds. The first step in the error correction is creating and verifying the 12-qubit cat state. Figure 18 shows how the ancillae "branches" are incorporated into the data branches. After verification, the cat state is moved to the appropriate data branches, where it is CNOT'ed with the data qubits. The cat state is then moved back and uncreated, while the data branches are error-corrected. Finally, a Hadamard is applied to the last cat-state ancilla, which is then redundantly measured. The layout in Fig. 18 is not necessarily optimal.

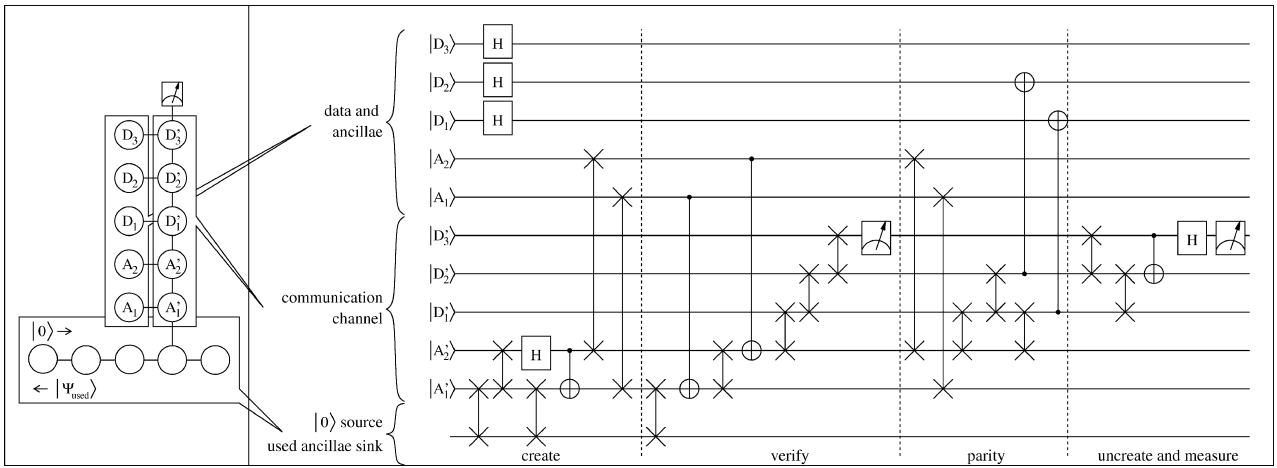


FIG. 17. "Two-rail" layout for the three-qubit phase-correction code. The schematic on the left shows qubit placement and communication, where D_i's indicate data qubits, and A_i's are cat-state ancillae. The column of D_i''s and A_i''s form a swapping channel, and can also interact with the data and cat-state ancilla. The open qubit swapping channel at the bottom brings in fresh ancillae, and removes used ancillae. The same layout is shown as a quantum circuit on the right, with the operations required to create and verify an ancillary cat state, and to measure the parity of a pair of data qubits.

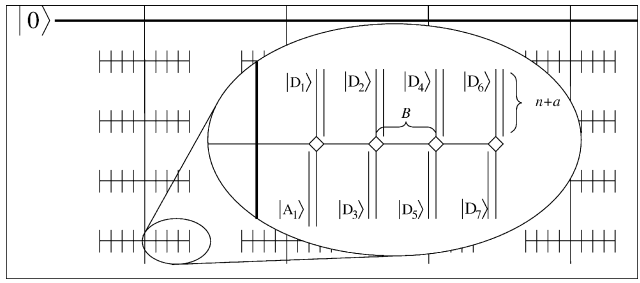


FIG. 18. Schematic layout of the H-tree structure of a concatenated code. The branches labeled $|D_i\rangle$ are for logical data qubits, and consist of two rails of eleven qubits each—seven qubits for data and four for ancillae. The branch labeled $|A_1\rangle$ is for creating, verifying, and uncreating the cat state.

For $[\![7, 1, 3]\!]$ concatenated with itself k times,

$$t_{\mathrm{cc},k} \approx \lceil \log_2(a_k) \rceil t_{\mathrm{cnot}} + \left(\frac{5}{2} a_k - 3\right) t_{\mathrm{swap}}, \tag{17}$$

$$t_{\mathrm{cv},k} = 2a_k t_{\mathrm{cnot}} + \big(a_k(a_k - 2) + 2\big) t_{\mathrm{swap}}, \tag{18}$$

$$t_{\mathrm{p},k} = a_k + 3t_{\mathrm{b},k} + 3t_{\mathrm{b},k-1} + t_{\mathrm{ecc},k-1}, \tag{19}$$

$$t_{\mathrm{cu},k} = t_{\mathrm{cc},k} + t_{\mathrm{single}} + t_m, \tag{20}$$

$$a_k = 4 \times 3^{k-1}, \quad \text{and} \tag{21}$$

$$t_{\mathrm{b},k} = \begin{cases} 1, & k = 1, \\ B, & k = 2, \\ t_{\mathrm{b},k-1} + (n + a_1) t_{\mathrm{b},k-2}, & k = 3, \\ t_{\mathrm{b},k-1} + 2\lceil n/2 \rceil t_{\mathrm{b},k-2}, & k > 3, \end{cases} \tag{22}$$

where the subscript k indicates the level of encoding, a_k is the number of qubits in the cat state at level k, $t_{\mathrm{b},k}$ is the branch distance between logical qubits at level k, B is the minimum number of qubits between two branches for a given architectural model, and n is the number of physical qubits in the non-concatenated code.

With communication by swapping channel, the SWAP operator becomes very important. In the sea-of-qubits model, SWAP's are not required. In the model described above, SWAP's account for over 80% of all operations.

7.3 Avoiding Correlated Errors

An important assumption in quantum error correction is that errors in the redundant qubits of a codeword are uncorrelated. That is, we do not want one error in a

codeword to make a second error more likely. To avoid such correlation, it is important to try not to interact qubits in a codeword with each other.

Unfortunately, we find that a 2D layout cannot avoid indirect interaction of qubits in a codeword. At some point, all the qubits in a codeword must be brought to the same physical location in order to calculate error syndromes. In order to do this, they must pass through the same line of physical locations. Although we can avoid swapping the codeword qubits with each other, we cannot avoid swapping them with some of the same qubits that flow in the other direction.

For concreteness, if two qubits of codeword d_0 and d_1 both swap with an ancilla a_0 going in the opposite direction, there is some probability that d_0 and d_1 will become correlated with each other through the ancilla. This occurs if both SWAPs experience a partial failure. In general, if p is the probability of a failure of a SWAP gate, the probability of an error from swapping a logical qubit is

$$n^k b_k p + \binom{n^k}{2} b_k p^2 + \binom{n^k}{3} b_k p^3 + \cdots,$$

where b_k is the number of qubits between branches at level k, and the higher order terms are due to correlation between the qubits. From this form, it is clear that correlated errors are dominated by uncorrelated errors, when $n^k p \ll 1$.

By calculating the number of basic computation and communication operations necessary to use teleportation for long-distance communication, we can quantify when we should switch from swapping to teleportation in our tree structure. Figure 19 illustrates this tradeoff. We can see that for $B = 22$, teleportation should be used when $k \geqslant 5$.

7.4 Teleportation

Table II lists the number of SWAP operations required to move an unencoded qubit from one level-k codeword to the adjacent codeword for different minimum branch distances, as well as the total operations to teleport the same qubit. Since a teleportation channel precommunicates EPR pairs, it has a fixed cost. To use teleportation for our circuit, we must evaluate the number of computation and communication operations within the teleportation circuit. By comparing this number of operations with the swapping costs from the previous section, we can decide at what level k of the tree to start using teleportation instead of swapping for communication.

Teleportation has another advantage, which is beyond the scope of this study. By suitably modifying the EPR pairs, teleportation can be used to perform operations at a distance [28]. It does not eliminate the need for error correction, and correctly modifying the EPR pairs has its own costs. This is an interesting area for future research.

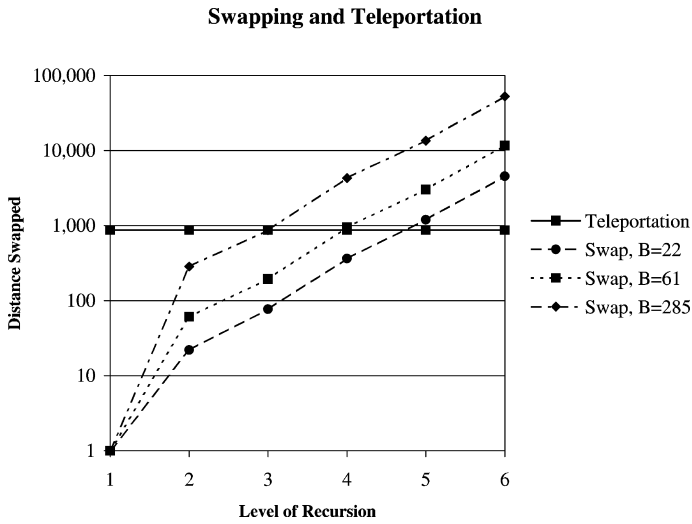


FIG. 19. Cost of teleportation compared to swapping. The B-values chosen illustrate break-even points for different levels of recursion.

TABLE II
COMPARISON OF THE COST OF SWAPPING AN ENCODED QUBIT TO THE COST OF TELEPORTING IT. THE B-VALUES ARE THE DISTANCE BETWEEN ADJACENT QUBITS

k	Teleportation	Swapping, $B = 22$	Swapping, $B = 61$	Swapping, $B = 285$
1	864	1	1	1
2	864	22	61	285
3	864	77	194	866
4	864	363	948	4, 308
5	864	1, 199	3, 032	13, 560
6	864	4, 543	11, 680	52, 672

8. System Bandwidth

Our goal has been to design a reliable, scalable quantum communication layer that will support higher-level quantum error correction and algorithms functioning on top of this layer. A key issue for future evaluation, however, is that the lower latency of our teleportation channel actually translates to even higher bandwidth when the upper layers of a quantum computation are considered. It is for this reason that long wires should not be constructed from chained swapping-channels and quantum "repeaters."

The intuition behind this phenomenon is as follows. Quantum computations are less reliable than any computation technology that we are accustomed to. In fact, quantum error correction consumes an enormous amount of overhead both in terms of redundant qubits and time spent correcting errors. This overhead is so large that the reliability of a computation must be tailored specifically to the run length of an algorithm. The key is that, the longer a computation runs, the stronger the error correction needed to allow the data to survive to the end of the computation. The stronger the error correction, the more bandwidth consumed transporting redundant qubits. Thus, lower latency on each quantum wire translates directly into greater effective bandwidth of logical quantum bits.

9. Conclusion

Quantum computation is in its infancy, but now is the time to evaluate quantum algorithms under realistic constraints and derive the architectural mechanisms and reliability targets that we will need to scale quantum computation to its full potential. Our work has focused upon the spatial and temporal constraints of solid-state technologies.

Building upon key pieces of quantum technology, we have provided an end-to-end look at a quantum wire architecture. We have exploited quantum teleportation to enable pipelining and flexible error correction. We have shown that our teleportation channel scales with distance and that swapping channels do not. Finally, we have discovered fundamental architectural pressures not previously considered. These pressures arise from the need to co-locate physical phenomena at both the quantum and classical scale. Our analysis indicates that these pressures will force architectures to be sparsely connected, resulting in coarser-grain computational components than generally assumed by previous quantum computing studies.

At the systems level, the behavior of wires becomes a crucial limiting factor in the ability to construct a reliable quantum computer from faulty parts. While the Threshold Theorem allows fault-tolerant quantum computers to be realized in principle, we showed that in practice many assumptions must be carefully scrutinized, particularly for implementation technologies that force a two-dimensional layout scheme for qubits and their interconnects. Our analysis suggests that, rather counterintuitively, fault-tolerant constructions can be more resource efficient than equivalent circuits made from more reliable components, when the failure probability is a function of resources required. And a detailed study of the resources required to implement recursive quantum error correction circuits highlights the crucial role of qubit communication, and in particular, the dominant role of SWAP gates. We find that at a

certain level of recursion, resources are minimized by choosing a teleportation channel instead of the SWAP. It is likely that the reliability of the quantum SWAP operator used in short-distance communication will be the dominant factor in future quantum architecture system reliability.

ACKNOWLEDGEMENTS

Thanks to Isaac Chuang, John Kubiatowicz, John Owens, Matt Farrens, Mark Whitney and Diana Keen for their helpful comments on preliminary material for this paper, and Andrew Cross for verifying the detailed quantum circuit constructions. This work is supported in part by the DARPA Quantum Information, Science and Technology Program, by NSF CAREER grants to Mark Oskin and Fred Chong, and by a UC Davis Chancellor's Fellowship to Fred Chong. Earlier portions of this work have appeared in conferences [36] and [57].

REFERENCES

[1] Lloyd S., "Quantum-mechanical computers", *Scientific American* **273** (October 1995) 44.
[2] Nielsen M., Chuang I., *Quantum Computation and Quantum Information*, Cambridge Univ. Press, Cambridge, UK, 2000.
[3] DiVincenzo D.P., "Quantum computation", *Science* **270** (5234) (1995) 255, quant-ph/9503016.
[4] Gershenfeld N., Chuang I., "Quantum computing with molecules", *Scientific American* (June 1998).
[5] Maklin Y., Schön, Schnirman A., "Quantum state engineering with Josephson-junction devices", *Rev. Modern Phys.* **73** (2) (2001) 357–400.
[6] Bennett C.H., DiVincenzo D.P., "Quantum information and computation", *Nature* **404** (2000) 247–255.
[7] Dykman M.I., Platzman P.M., Seddighrad P., "Qubits with electrons on liquid helium", *Phys. Rev. B* **67** (2003) 155403.
[8] Vandersypen L.M., Steffen M., Breyta G., Yannoni C.S., Cleve R., Chuang I.L., "Experimental realization of order-finding with a quantum computer", *Phys. Rev. Lett.* **85** (December 15, 2000) 5453–5455.
[9] Knill E., Laflamme R., Martinez R., Tseng C., "An algorithmic benchmark for quantum information processing", *Nature* **404** (March 2000) 368–370.
[10] Sackett C., Kielpinsky D., King B., Langer C., Meyer V., Myatt C., Rowe M., Turchette Q., Itano W., Wineland D., Monroe C., "Experimental entanglement of four particles", *Nature* **404** (2000) 256–258.
[11] Leibfried D., Blatt R., Monroe C., Wineland D., "Quantum state engineering with Josephson-junction devices", *Rev. Modern Phys.* **75** (January 2003) 281–324.

[12] Kane B., "A silicon-based nuclear spin quantum computer", *Nature* **393** (1998) 133–137.
[13] Vion D., Aassime A., Cottet A., Joyez P., Pothier H., Urbina C., Esteve D., Devoret M.H., "Manipulating the quantum state of an electrical circuit", *Science* **296** (2002) 886.
[14] Kielpinsky D., Monroe C., Wineland D., "Architecture for a large-scale ion trap quantum computer", *Nature* **417** (2002) 709.
[15] Bennett C.H., Brassard G., Crépeau C., Jozsa R., Peres A., Wootters W., "Teleporting an unknown quantum state via dual classical and EPR channels", *Phys. Rev. Lett.* **70** (1993) 1895–1899.
[16] Nielsen M.A., Chuang I.L., *Quantum Computation and Quantum Information*, Cambridge Univ. Press, Cambridge, UK, 2000.
[17] Black P.E., Kuhn D.R., Williams C.J., "Quantum computing and communication", in: *Advances in Computes*, vol. 56, 2002.
[18] Grover L., in: *Proc. 28th Annual ACM Symposium on the Theory of Computation, New York*, Assoc. Comput. Mach. Press, 1996, pp. 212–219.
[19] Shor P., "Algorithms for quantum computation: Discrete logarithms and factoring", in: *Proc. 35th Annual Symposium on Foundations of Computer Science, Los Alamitos, CA*, IEEE Press, 1994, p. 124.
[20] Shor P.W., "Scheme for reducing decoherence in quantum computer memory", *Phys. Rev. A* **54** (1995) 2493.
[21] Calderbank A., Shor P., "Good quantum error-correcting codes exist", *Phys. Rev. A* **54** (1996) 1098.
[22] Steane A., "Multiple particle interference and quantum error correction", *Proc. R. Soc. London A* **452** (1996) 2551–2576.
[23] Laflamme R., Miquel C., Paz J.-P., Zurek W.H., "Perfect quantum error correction code", *Phys. Rev. Lett.* **77** (1996) 198, quant-ph/9602019.
[24] Steane A., "Simple quantum error correcting codes", *Phys. Rev. Lett.* **77** (1996) 793–797.
[25] Gottesman D., "A class of quantum error-correcting codes saturating the quantum hamming bound", *Phys. Rev. A* **54** (1996) 1862–1868.
[26] Calderbank P.S.A., Rains E., Sloane N., "Quantum error correction and orthogonal geometry", *Phys. Rev. Lett.* **78** (1997) 405–409.
[27] Rivest R.L., Shamir A., Adleman L.M., "A method for obtaining digital signatures and public-key cryptosystems", Tech. Rep. MIT/LCS/TM-82, MIT, 1978.
[28] Gottesman D., Chuang I.L., "Quantum teleportation is a universal computational primitive", *Nature* **402** (1999) 390–392.
[29] Monroe C., Meekhof D.M., King B.E., Itano W.M., Wineland D.J., "Demonstration of a fundamental quantum logic gate", *Phys. Rev. Lett.* **75** (1995) 4714.
[30] Turchette Q.A., Hood C.J., Lange W., Mabuchi H., Kimble H.J., "Measurement of conditional phase shifts for quantum logic", *Phys. Rev. Lett.* **75** (1995) 4710.
[31] Vandersypen L.M., Steffen M., Sherwood M., Yannoni C., Breyta G., Chuang I.L., "Implementation of a three-quantum-bit search algorithm", *Appl. Phys. Lett.* **76** (5) (2000) 646–648.
[32] Yu Y., Han S., Chu X., Chu S.-I., Wang Z., "Coherent temporal oscillations of macroscopic quantum states in a Josephson junction", *Science* (May 2002) 889–892.

[33] Coffey M.W., "Quantum computing based on a superconducting quantum interference device: Exploiting the flux basis", *J. Modern Optics* **49** (14) (2002) 2389–2398.
[34] Vrijen R., Yablonovitch E., Wang K., Jiang H.W., Balandin A., Roychowdhury V., Mor T., DiVincenzo D., "Electron spin resonance transistors for quantum computing in silicon–germanium heterostructures", quant-ph/9905096, 1999.
[35] Skinner A., Davenport M., Kane B., "Hydrogenic spin quantum computing in silicon: A digital approach", *Phys. Rev. Lett.* **90** (February 2003).
[36] Oskin M., Chong F., Chuang I., Kubiatowicz J., "Building quantum wires: The long and the short of it", in: *Proc. International Symposium on Computer Architecture (ISCA 2003), New York*, Assoc. Comput. Mach. Press, 2003.
[37] Kane B., McAlpine N., Dzurak A., et al., "Single-spin measurement using single-electron transistors to probe two-electron systems", *Phys. Rev. B* **61** (January 2000) 2961–2972.
[38] Globus A., Bailey D., Han J., Jaffe R., Levit C., Merkle R., Srivastava D., "NASA applications of molecular nanotechnology", *J. British Interplanetary Society* **51** (1998).
[39] Adleman L., "Toward a mathematical theory of self-assembly", USC Tech. Report, 2000.
[40] Anderson E., Boegli V., Schattenburg M., Kern D., Smith H., "Metrology of electron beam lithography systems using holographically produced reference samples", *J. Vac. Sci. Technol.* **B-9** (1991) 3606.
[41] Sanie M., Cote M., Hurat P., Malhotra V., "Practical application of full-feature alternating phase-shifting technology for a phase-aware standard-cell design flow", in: *Design Automation Conference (DAC)*, 2001, pp. 93–96.
[42] Ferry D.K., Goodnick S.M., *Transport in Nanostructures*, in: *Cambridge Studies in Semiconductor Physics & Microelectronic Engineering*, vol. 6, Cambridge Univ. Press, Cambridge, UK, 1997.
[43] Likhareve K.K., "Single-electron devices and their applications", in: *Proceedings of the IEEE*, vol. 87, 1999, pp. 602–632.
[44] Tucker J.R., Shen T.-C., "Can single-electron integrated circuits and quantum computers be fabricated in silicon?", *Internat. J. Circuit Theory and Applications* **28** (2000) 553–562.
[45] Oskin M., Chong F., Chuang I., "Overhead reduction in a architecture for quantum computers", *IEEE Computer* **35** (1) (2002) 79–87.
[46] Kane B.E., McAlpine N.S., Dzurak A.S., Clark R.G., Milburn G.J., Sun H.B., Wiseman H., "Single spin measurement using single electron transistors to probe two electron systems", Phys. Rev. B, submitted for publication, cond-mat/9903371, 1999.
[47] Aharonov D., Ben-Or M., "Fault tolerant computation with constant error", in: *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, 1997, pp. 176–188.
[48] Knill E., Laflamme R., Zurek W.H., "Resilient quantum computation", *Science* **279** (5349) (1998) 342–345, quant-ph/9702058.
[49] Preskill J., "Fault-tolerant quantum computation", in: Lo H.-K., Spiller T., Popescu S. (Eds.), *Quantum Information and Computation*, World Scientific, Singapore, 1998.
[50] Verhulsta A., et al., "Non-thermal nuclear magnetic resonance quantum computing using hyperpolarized xenon", *Appl. Phys. Lett.* **79** (15) (2001).

[51] Schulman L.J., Vazirani U., "Scalable NMR quantum computation", quant-ph/9804060, 1998. Extended abstract appears in 31st Annual ACM Symp. on Theory of Computing (STOC).

[52] Schulman L.J., Vazirani U., "Molecular scale heat engines and scalable quantum computation", in: *Proc. 31st Ann. ACM Symp. on Theory of Computing (STOC '99)*, 1999, pp. 322–329.

[53] Bennett C.H., DiVincenzo D.P., Smolin J.A., Wootters W.K., "Mixed state entanglement and quantum error correction", *Phys. Rev. A* **54** (1996) 3824, quant-ph/9604024.

[54] Bennett C.H., Bernstein H.J., Popescu S., Schumacher B., "Concentrating partial entanglement by local operations", *Phys. Rev. A* **53** (4) (1996) 2046–2052, quant-ph/9511030.

[55] Bennett C.H., Brassard G., Popescu S., Schumacher B., Smolin J.A., Wootters W.K., "Purification of noisy entanglement and faithful teleportation via noisy channels", *Phys. Rev. Lett.* **76** (1996) 722, quant-ph/9511027.

[56] Gottesman D., "Fault tolerant quantum computation with local gates", quant-ph/9903099, 1999.

[57] Copsey D., et al., "The effect of communication costs in solid-state quantum computing architectures", in: *Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA 2003), New York*, Assoc. Comput. Mach. Press, 2003.

Author Index

Numbers in *italics* indicate the pages on which complete references are given.

T

Subject Index

Q

This Page Intentionally Left Blank

Contents of Volumes in This Series

Volume 43

Volume 44

Volume 45

Volume 46

Volume 47

Volume 48

Volume 49

Volume 50

Volume 51

Volume 52

Volume 53

Volume 56

Volume 57

Volume 58