

Cooperative Solutions to the Dynamic Management of Communication Resources

A Thesis
Presented to
The Academic Faculty

by

Robin H. Kravets

In Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Georgia Institute of Technology
July 29, 1998

Cooperative Solutions to the Dynamic Management of Communication Resources

Approved:

Dr. Ken L. Calvert
(College of Computing)

Dr. Karsten Schwan
(College of Computing)

Dr. P. Krishnan
(Bell Labs)

Dr. Mustaque Ahamad
(College of Computing)

Dr. Don Towsley
(University of Massachusetts, Amherst)

Date Approved _____

Acknowledgments

First of all I would, I would like to thank my advisors Ken Calvert and Karsten Schwan. Between the three of us, I believe we not only spanned the area between networking and systems, but also the distance between the CoC and GCATT. I would like to thank P. Krishnan for his guidance during my time at Bell Labs and his continuing help after I left. I would also like to thank Mustaque Ahamad and Don Towsley for all of their help and guidance during this last phase of my research.

There are many students who have helped me with my research and helped me simply get by at Georgia Tech. I would especially like to thank Don Allison (now I know the resistance of a jelly donut), Colleen Kehoe, Fabian Bustamante, Dave Brogan, Josh Berman, Vernard Martin, Wayne Wooten and Anind Dey. Extra special thanks to the Happy Hour Crowd, without whom I never would have made it through my first year. I wish everyone success, however they may find it. I would also like to thank the faculty and staff of the College of Computing. My special thanks to Jalisa Norton, Amy Bruckman, Ellen Zegura, Jessica Hodgins, Beth Mynatt and Blair MacIntyre.

My friends have stood by and supported me from all over the country and all over the world. If I started to name them all, my thesis would be too long. Thanks everyone!

My family has believed in me through three universities and three degrees. I know they are proud to have a new “doctor” in the family. Thanks to Dad, Susie, Marc and especially Mom. Special thanks to Milo Dundon, who first drew me into the world of computing and has believed in me ever since.

But most of all, I would like to give my deepest and dearest thanks to Rob Kooper. His support and persistence helped me through many tough times. And his intelligence helped me with many tough problems. Rob, I thank you from the bottom of my heart.

Contents

Acknowledgments	iii
List of Tables	vi
List of Figures	viii
Summary	xi
Chapter	
1 Introduction	1
1.1 Resource Specification and Management	2
1.2 Approach	4
1.2.1 Motivating Applications	4
1.2.2 Motivating Environments	4
1.2.3 Communication Framework	5
1.2.4 Adaptive Protocols	5
1.2.5 Adaptation Techniques	5
1.3 Thesis Outline	6
2 Problem Domain and Description	7
2.1 Communication Layer	7
2.2 Application Service Specification	9
2.3 Communication Resource Specification	11
2.4 Communication Adaptation	12
3 Dynamic Communication Configuration Framework	15
3.1 Example Application	15
3.2 Framework Design	16
3.3 Background and Related Work	16
3.4 The Configurable Protocol Subsystem	17
3.4.1 Configurations	18
3.4.2 Protocol Infrastructure	18
3.4.3 Metaheader Protocol	19
3.4.4 Protocol Functions	20
3.5 Interfaces and Implementation	20
3.5.1 Application to Protocol Infrastructure Interface	20
3.5.2 Application to Protocol Functions Interface	22
3.5.3 Protocol Infrastructure to Protocol Functions Interface	22
3.6 Runtime Configuration	23
3.7 Overview of Contributions	23

4	Mobile Communication and Power Management	25
4.1	Power Management for Mobile Hosts	25
4.2	Communication Model and Power Management	27
4.3	Communication-Based Power Management	28
4.3.1	Power Management Control Protocol	29
4.3.2	Timing Considerations	31
4.4	Experimental Evaluation	32
4.4.1	Energy Measurement	32
4.4.2	Experimental Setup	34
4.4.3	Communication Patterns	36
4.4.3.1	Simulation-based Communication Patterns	36
4.4.3.2	Trace-Based Communication Patterns	36
4.4.4	Results	37
4.4.4.1	Protocol Energy Consumption	37
4.4.4.2	Energy Savings for Communication Patterns	39
4.4.4.3	Delay	40
4.4.5	Impact on System Costs	42
4.5	Adaptive Mobile Power Management	43
4.6	Overview of Contributions	44
5	Variable Reliability Protocol	47
5.1	Reliability and Communication	47
5.2	Specification of Variable Reliability without Timing Constraints	48
5.2.1	Loss Tolerance	48
5.2.2	Message Order	49
5.2.3	Application-Specified Data Boundaries	50
5.2.4	Protocol Specification	50
5.3	Variable Reliability Protocol	52
5.3.1	Implementation	52
5.3.1.1	Protocol State Block	52
5.3.1.2	Example	54
5.3.2	Experimental Results	57
5.4	Overview of Contributions	58
6	Payoff-Based Adaptation	59
6.1	Target Application	59
6.2	Adaptive Quality Specification	60
6.2.1	Locality-Based Adaptive Quality Specification	61
6.2.2	Distance-Based Adaptive Quality Specification	62
6.2.3	Initial Conclusions	63
6.3	Payoff Adaptation with Observed Network Behavior	63
6.4	Payoff Adaptation with Loss-Load Curves	66
6.5	Overview of Contributions	70

7	Power-Aware Communication for Mobile Computing	71
7.1	Mobility, Communication and Power Management	72
7.1.1	The Mobile Environment	73
7.1.2	Energy Consumption Model	73
7.2	Power-Aware Communication	74
7.3	Experimental Evaluation	74
7.3.1	Energy Consumption for TCP	75
7.3.2	Energy Consumption for SACK	76
7.4	Towards Adaptive Power Management	79
7.4.1	Optimizing Energy Consumption	79
7.5	Conclusions	81
8	Contributions and Future Directions	83
8.1	Future Directions	84
A	Configurable Protocol Engine	87
A.1	Data Structures	87
A.2	Functions	87
A.2.1	Sending Data	87
A.2.2	Receiving Data	89
B	Power Control Protocol	93
C	Power Measurement Details	95
D	Variable Reliability Protocol	97
D.1	Sending Protocol	97
D.2	Receiving Protocol	99
	Bibliography	104
	Vita	111

List of Tables

1	Power Requirements of the Lucent WaveLAN PCMCIA Wireless Ethernet card. . .	35
2	Data Communication Patterns for Three Simulated Users	37
3	Timing Patterns for Three Simulated Users	37
4	Measured Power Requirements for Three Machines	42
5	Average Updates Received and Messages Retransmitted (User 0)	63
6	Results for Amount of Data Received and Transfer Time	69

List of Figures

1	Communication Architecture	8
2	Sample Payoff Functions for Reliability	10
3	Sample Payoff Functions for Time	11
4	Example Loss-Load Curve	12
5	Optimization Algorithm	13
6	Multimedia Slideshow	16
7	Architecture of Protocol Subsystem	19
8	Example Message using Metaheader Protocol	20
9	Interfaces between Application, Protocol Infrastructure, Protocol Functions, and Network	21
10	Slave (Base Station) Protocol State Diagram	30
11	Master (Mobile Host) Protocol State Diagram	31
12	Energy Components	33
13	Experimental Setup	34
14	Sample Output from Multimeter	35
15	Trace-Based Data Usage	38
16	Power Readings During a Sample Idle Period	38
17	Energy Savings During Idle Periods	39
18	A Sample Transmission	39
19	Percent Savings for Communication Energy Consumption for Simulation-Based Communication Patterns	40
20	Percent Savings for Communication Energy Consumption for Trace-Based Communication Patterns	40
21	Average Added Delay based on Sleep Duration	41
22	Percent Savings for Three Types of Machines	43
23	Percent of Total Time spent Sleeping	44
24	Example of Allowable Loss in a Sliding Window and in a Data Frame	49
25	Example of History Window and Reliability Window	54
26	Example of Variable Reliability Protocol – Step 1	54
27	Example of Variable Reliability Protocol – Step 2	55
28	Example of Variable Reliability Protocol – Step 3	55
29	Example of Variable Reliability Protocol – Step 4	55
30	Example of Variable Reliability Protocol – Step 5	56
31	Example of Variable Reliability Protocol – Step 6	56
32	Example of Variable Reliability Protocol – Step 7	56
33	Example of Variable Reliability Protocol – Step 8	56
34	Example of Variable Reliability Protocol – Step 9	57
35	Example of Variable Reliability Protocol – Step 10	57
36	Example of Variable Reliability Protocol – Step 11	57
37	Transmission Time	58
38	Image with Application-Specified Quality Regions	61
39	Transfer Time for Quality-Based Partitioned Image	62

40	Snapshot of Distributed Virtual Environment Application with Variable Reliability Rings	63
41	Payoff Function for Reliability	65
42	Payoff Function for Time	65
43	Final Payoff	65
44	Time Statistics	66
45	Data Statistics	66
46	Payoff Function for Reliability	67
47	Net Payoff Curves	68
48	Observed Loss over Time	68
49	Application Reliability and Bandwidth over Time	69
50	Total Payoff over Time	69
51	Energy Consumption for TCP	76
52	Running Time	77
53	Unnecessary Retransmissions	77
54	Energy Components	78
55	Total Energy Consumption	78
56	Algorithm for Determining Minimum Energy Consumption for Data Transmissions using FEC	80
57	Diagram of Power Measurement Setup	95

Summary

Support for large-scale distributed interactive applications demands resource management infrastructures that understand the relationship between application requirements and the services available from the network. To provide such support, we have designed and implemented an end-to-end communication layer which mediates between an application and a network service. By using configurable communication protocols, this layer can adaptively configure an application's communication parameters as demanded by changes in both service requirements and resource availability. Adaptations maximize the perceived benefits a specific communication configuration offers to the application, by permitting applications to provide dynamic service quality requirements to the communication layer in the form of "payoff functions". As a result, the communication layer supports precisely the cost-quality tradeoffs the user is willing to make.

The effectiveness of these techniques is demonstrated with two different cost-quality tradeoff scenarios. The first scenario explores the use of a variable reliability protocol that realizes reduced transfer delays by exploiting tradeoffs with respect to the reliability of message communication. The second scenario involves power management for mobile computers, including techniques for saving energy during active communication, as well as during idle periods. The communication layer exploits certain aspects of mobile communication protocols to understand the tradeoffs between energy consumption and transfer delays.

Chapter 1

Introduction

The need for intelligent resource management is going to increase as we see an increase in the demand for large-scale distributed interactive applications. We already see the growth of such applications in many diverse environments, from distributed interactive simulations being developed for use across the Internet to computer supported cooperative work environments for use in mobile environments. We also see the demand for resources to the home increasing far more quickly than the availability of those resources. All of these scenarios share the problem of limited and dynamically changing resource availability. Our research aims to support distributed applications operating in these dynamic environments. The diversity of the requirements of such applications demands that we provide resource management infrastructures that understand the mechanisms of resource management, but leave the policy decisions up to the specific applications in question.

The goal of this research is to address issues in resource management for a large class of distributed multimedia applications, including video conferencing systems, web applications, distributed interactive simulations and distributed virtual environments. In this context, intelligent resource management is important due to the dynamic behavior of such applications and their operation in network environments like the Internet and wireless networks, both of which offer dynamically changing and often unpredictable network service. Ideally, intelligent resources management is realized through the adaptation of application behavior in response to changes in network conditions. Adaptations are based on information about the availability of current and future network services. In addition, network services are configured to match the requirements of individual applications.

The problem addressed by this research is intelligent resource management, as described above. Our approach and solution utilize a communication layer that is guided by both the application and the network. The communication layer's interface to the application hides network- and system-specific details from the application. The application need only supply information about its requested service quality, as well as any changes to its requested service quality. The abstraction provided by the interface between the application and the communication layer allows the application to define service quality in terms of parameters that are meaningful to it. These parameters range from application-specific notions of data quality, such as image quality or delays in data transfers, to system- or machine-specific notions of resource usage, such as CPU usage or available energy in a mobile host. The communication layer's interface to the network hides application-specific details from the network. The network supplies service availability information that enables the communication layer to make changes in the communication services offered to the application in accordance with changes in service availability from the network. This mapping from network service availability to application service requests is realized through the use of configurable communication protocols and protocol-level knowledge of service quality. Configurable

protocols with which we experiment include power management protocols, variable reliability protocols and power-aware transmission protocols.

The key innovative contribution of our work is its cooperative solution to dynamic resource management, implemented by a communication layer that uses both application resource requirements and network resource availability to determine how to best configure the application's communications. The communication protocols are configured using application-specific service quality requirements stated in the form of *payoff functions*. These functions enable the communication layer to assess the benefits of certain configurations without additional interactions with the application. In its configuration decisions, the communication layer uses information about network service in the form of *service availability curves*. Operating points and updates for these curves are provided to the communication layer by a network agent. This information can be captured by monitoring and analyzing current traffic patterns.

Present distributed applications are not typically designed to adapt to changes in network conditions. To compound the problem, without information about the state of the network, there is little an application can do to compensate for the effect of changes in the service it receives. At the same time, network-based adaptation does not typically consider the requirements of specific applications and, therefore, has limited opportunities to improve the match of application requirements with available network resources. Furthermore, it has been argued that network-level resource management must be aimed at benefiting all applications [59]. Such an argument discourages specific network-level adaptations for individual applications or application classes. In response to these arguments, we proposed the use of our cooperative communication layer, which has no expectations of application knowledge of network behavior or network knowledge of application behavior.

The advantages of using our cooperative communication layer as an intermediary between applications and the network are diverse. First, application-level runtime adaptation may be implemented using abstractions meaningful to the application, rather than in terms of network- or protocol-specific parameters. Such abstractions facilitate application implementation and also isolate applications from undue knowledge about the network and its protocols. Second, service adaptations performed in the communication layer may take into account the specific needs of applications while also taking advantage of knowledge about the availability of network services. The adaptations are realized through the use of payoff functions, through interactions with network protocols implemented within the communication layer, and through the use of knowledge about network services based on service availability curves. Third, service availability curves constitute a general and portable mechanism for capturing the past, current and future behavior of network services.

1.1 Resource Specification and Management

Resource management may be decomposed into three topics. The first topic concerns techniques for specifying resource needs or availability. If the resource in question is network bandwidth, we will need to specify how much bandwidth the application needs as well as how much bandwidth is available from the network. The second topic concerns the ability, or lack thereof, to monitor and react to changes in existing specifications. For example, if we are trying to conserve battery lifetime in a mobile host, it may be beneficial to behave differently if there is a fully-charged battery, as compared to a battery that is almost empty. This is only possible if the resource management techniques being used are designed to understand the fact that the state of the resource has changed. The third topic concerns the scope of the resource management techniques. Simple

resource management aimed at a single resource has a very limited scope, while more complex resource management that considers multiple resources has a broader scope. If the scope of a resource management technique is too limited, it may reduce its effectiveness. On the other hand, if the scope is too broad, the techniques may become too complex. The rest of this section discusses these three topics and compares research in related areas to the approaches taken in our research.

In the context of networking resources, a variety of techniques have been employed for resource specification in multimedia or real-time applications. Typically, these techniques assume that applications state their resource requirements as specific levels of service (e.g., as in RSVP [69]) or as feasible service ranges [8] or regions [70]. In comparison, our approach states application needs and service availability in functional forms, using payoff functions and service availability curves. As an argument for the utility of functional resource specification, consider a resource reservation scheme that provides the application with a simple yes/no answer as to whether some desired level of service may be met. In this example, the application has to probe the reservation agent multiple times to determine some feasible level of service. By increasing the richness of this service interface, a more detailed response may be given, possibly involving feasible service ranges or even tradeoffs in the effective service the network can provide to the application based on the network-level resources utilized.

Once resource specifications have been stated, they may be employed at initialization time by resource reservation techniques like bandwidth reservation or fair share transmission scheduling [21]. However, as with complex real-time applications [62], a distributed multimedia application can rarely accurately predict its resource requirements for extended periods of time. This results in either inadequate reservations or in poor resource utilization throughout the application's execution. The adaptive solutions promoted in our work may be used throughout the execution of a distributed, interactive application. Consequently, the decisions being made are based on up-to-date information on resource availability and on application needs. Sample adaptation techniques include runtime task scheduling in real-time systems [62], load balancing or migration for real-time [56] or scientific [58] applications, and application-level adaptation in distributed [50] or parallel [11] programs.

The benefits of using the payoff-based adaptation techniques described in this thesis are derived from the richness of the information provided by payoff functions to the communication layer. The communication layer is given a wide range of resource allocation choices that will satisfy the application, and it has the payoff information necessary to determine operating points within this range that best benefit the application. In comparison, adaptation schemes that use specification ranges or regions treat all operating points within those ranges or regions as equivalent. As an example, consider a resource reservation scheme based on statistical reservation. The communication layer would be provided with a more useful picture of resource availability if such a scheme could indicate to the communication layer that it could provide a lower bandwidth service with a low probability of any loss all the way up to a high bandwidth service with a high probability of loss. From this type of information, the communication layer would be able to determine what kind of tradeoffs to make and the benefits that would result. These tradeoffs become the key to effectively using the resources available to the application. By providing mechanisms that will allow the communication layer to evaluate current resource availability, the communication layer can adapt to suit the requirements of the application.

The specification techniques we employ are shown generally useful in this thesis by their use for the management of different types of resources. This includes traditional networking resources, such as bandwidth, as well as more system-oriented resources, such as battery lifetime. In addition, we show how multiple resources may be managed, by embedding our techniques in our cooperative communication layer. In the context of power management for mobile computers, for instance,

traditional approaches focus on the energy consumption of individual devices such as disks [25, 30, 44], CPUs [28, 45, 67] and communication devices [65, 17, 64, 72]. Our techniques permit the incorporation of information about individual devices into a more complete power management solution.

1.2 Approach

This research supports distributed interactive applications, by providing an adaptive communication layer that configures communication protocols based on the benefits the communication layer expects to realize for the application. The key to such support is an understanding of the possible tradeoffs in communication and the ways in which these tradeoffs benefit the application. In an effort to allow these applications to continue operating in less than ideal situations, we also investigate techniques for allowing applications to adapt to the current state of resource availability. Our techniques are built upon four basic tools: application requirements specification with payoff functions, network availability specification with service availability curves, a configurable communication framework, and configurable communication protocols. With these tools, the communication layer can adapt to changes in network resource availability and application resource requirements, within the boundaries of the configurable protocols provided by the communication framework.

1.2.1 Motivating Applications

The driving applications for our work are derived from many different areas including, but not limited to, video conferencing systems [3], web applications [7], distributed interactive simulations [31], and distributed virtual environments [51]. The service requirements of such applications range from offering transfer times matching human perceptual needs (as in distributed virtual environments) to providing suitable reliability at acceptable speeds (as in medical imaging). In addition, any single application is likely to communicate via multiple data types, resulting in the simultaneous specification of multiple service requirements. The dynamic nature of these applications makes it difficult to quantify a priori the network services required during execution. With the goal of addressing this problem, we have designed and implemented three such applications: (1) a multimedia slideshow, (2) a distributed robot simulation and (3) a distributed virtual environment application.

1.2.2 Motivating Environments

Our research targets communication environments with limited and dynamic resources. A limited resource may be the energy capacity of a mobile host's battery, or the bandwidth available from the network. Dynamic resource behavior may be due to transitional changes like battery depletion or transmission failure in wireless systems, or it may be caused by interference or energy usage by other applications. Additionally, the usefulness of these resources will be affected by such factors as network loss, delay and jitter. In order to explore such scenarios, we focus on applications operating over the Internet and over wireless networks. Both of these platforms present distinct flavors of resource limitations that change dynamically over time. The applications described above were implemented in both Internet and wireless scenarios.

1.2.3 Communication Framework

The *dynamically configurable communication framework* provides an environment in which applications can configure and reconfigure communication as demanded by changes in both resource requirements and availability. The framework design has three major components. The configurable protocol subsystem provides the ability to reconfigure communication protocols at runtime without requiring connection tear-down and reestablishment. The interface to the application enables the runtime specification of application resource requirements in the form of payoff functions for multiple service parameters. The interface to the network enables the runtime specification of service availability information or monitoring information from the network. The framework has been designed and implemented and is used as a communication base in all of our experiments.

1.2.4 Adaptive Protocols

Within the context of our dynamically configurable communication framework, we have designed and implemented three communication protocols. Each of these protocols was designed with the ability to adaptively change protocol parameters at runtime. The diversity of these protocols demonstrates the flexibility of our resource management techniques.

- The *power management protocol* provides mechanisms for managing and reducing the energy consumption of a communication device during idle periods in the communication patterns of a mobile host. By monitoring the communication usage patterns of the application, we devise a policy for power management that reduces the total energy consumed by the communications and the mobile host. We demonstrate the effects of this protocol within the context of user communication patterns simulating web users, shared development environment users and e-mail users.
- The *variable reliability protocol* provides a framework in which each application can define its own notion of “reliability” for the transport of its data. The variable reliability framework can concurrently maintain multiple reliability streams, thereby allowing applications to switch between different reliability levels depending on the specific requirements of individual data items. This functionality provides the “dials” which can be adjusted in order to specify different levels of reliability when adapting communications. We demonstrate the effects of using this protocol through experiments within the context of the multimedia slideshow application and the distributed robot simulation.
- The *selective acknowledgment transmission protocol (SACK)* provides reliable data transmission by aggressively telling the sender which messages have been received and which messages can be assumed lost or reordered. Our extended SACK protocol has two operating modes from which it can choose. In standard mode, the protocol reacts normally to losses by retransmitting the lost message. In extended mode, the protocol reacts to losses by retransmitting the lost message twice. By monitoring the losses seen on the communication channel and understanding the effects of such losses on each operating mode, we can determine the amount of energy consumed by the transfer of data in each mode. We demonstrate the effectiveness of such techniques in the context of our image transfer application.

1.2.5 Adaptation Techniques

Payoff-based adaptation configures communications during application execution, by optimizing the use of available communication resources. We present the effects of using payoff adaptation in

the context of a single stream of data within an application. Although the application may have multiple data streams, the means for determining how to adapt are limited to knowledge about an individual stream. Payoff-based adaptation expands on the explicit adjustment of communication parameters described above, and provides richer interfaces by which the communication layer can receive information which can be used to adjust communication parameters on behalf of the application. Through experimentation, we demonstrate the effects of optimizing for two sets conflicting parameters: power and transfer time, and transfer time and data quality.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2, we discuss our communication layer and present the service management problems addressed by our research. We also present *payoff functions*, a mechanism for specifying value-based application service requirements, and *service availability curves*, a technique for specifying dynamic network resource availability. Chapter 3 discusses our configurable communication framework. Chapter 4 describes our mobile power management protocol which provides the ability to adapt communication to reduce extraneous energy consumption during idle periods in communication. Chapter 5 describes the design and implementation of a novel variable reliability protocol and presents the validation of the protocol through experimentation with trading reliability for reduced latency. In Chapter 6, we present the validation of payoff-based adaptation in the context of an image transfer application. We also discuss the effect of service availability curves in the form of loss-load curves. In Chapter 7, we return to the concept of power management and present techniques for reducing energy consumption during active communication through the use of intelligent transmission protocols.

Chapter 2

Problem Domain and Description

Our work addresses the problem of applications with dynamic resource requirements operating in environments with dynamic service behavior. The complexity of such a problem lies in the interrelationship between the resources required by the application and the services available from the network. Although usage and availability may change independently, it is often the case that changing the application's usage changes the service availability. For example, network loss may change independently from the bandwidth being used by the application. However, if the application changes its bandwidth request, network loss may be affected. Although it is in the application's best interest to know this information, many applications may not have access to or may not even be able to handle such information. The cooperative communication layer designed in our research acts as an intermediary between the application and the network. It shields the application and the network from having to know too much about each other. It also offers well-defined interfaces between the communication layer and both the application and the network.

In an effort to support dynamic applications, we have developed techniques for application requirement and network service specification. These specification techniques are the basis for dynamically adapting to changes in application resource requirements and network service availability. Adaptation decisions result from knowledge about the tradeoffs between the levels of service provided to the application across multiple services, trading, for example, reliability for delay or energy for delay. These tradeoffs are realized through runtime reconfiguration of the communication protocols used during data transfer. By dynamically configuring applications, we are adopting the typical application model used by a growing set of researchers and developers concerned with attaining high levels of service quality in the presence of resource limitations.

This chapter first introduces our cooperative communication layer, which provides the framework for our adaptation techniques. Next, we present *payoff functions*, a service specification technique which enables the application to place its own quantitative *value* on selected communication parameters. By using payoff functions in conjunction with dynamic resource information described by *service availability curves*, tradeoffs may be made across competing communication parameters. Benefits are realized through runtime reconfiguration of an application's communication parameters. Although this work aimed at a large set of adaptive communication protocols, the examples in this chapter focus on reliability of data transfer and its effect on transfer time. In Chapters 4 and 7, we will discuss similar resource management approaches in the context of power management for mobile communication.

2.1 Communication Layer

Our approach defines the architecture of an end-to-end *communication layer* which mediates between an application and a network service, as depicted in Figure 1. The communication layer

provides a mapping from the services available from the network to the services required by the application. The application submits data units to the end-to-end communication layer for transmission; the communication layer processes them and then interacts with the network service to transfer the information through the network. The communication layer may then process the transferred information further on the receiving side before ultimately passing data units up to the application. Thus, the communication layer implements a channel using the given network service, and makes it available to the application. The characteristics of this channel—as quantified by measures such as throughput, delay, delay jitter, loss rate, and cost, and by semantic characteristics such as order preservation—determine its *value* to the application.

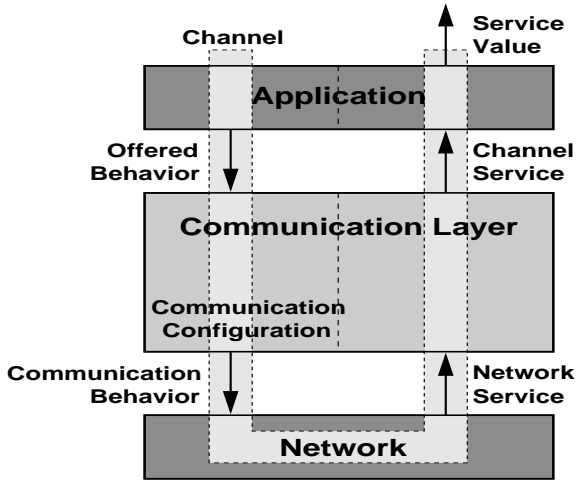


Figure 1: Communication Architecture

Our goal is to design the communication layer to adapt its behavior to maximize service value in the face of changing application needs and network characteristics. We can view this as an optimization problem, the structure of which is depicted in Figure 1. The communication layer has access to several sets of interrelated variables, some of which it can control. The relevant sets of variables are:

- Offered Behavior:** which traditionally is quantified by parameters such as rate and burstiness of data generation, but may also be characterized by information about the type and relative importance of data units being transferred. The application controls the offered behavior.
- Communication Configuration:** which captures the set of protocols and mechanisms used by the communication layer to enhance the service received from the network. For example, this variable captures whether forward error control or retransmission is used for error recovery. The communication layer controls the communication configuration directly. Controlled by the communication layer.
- Communication Behavior:** what the network “sees” from the communication layer. This includes traditional QoS dimensions such as peak and mean transmission rate, burst length, etc. The communication layer controls the communication behavior directly.
- Network Service:** what the communication layer “sees” from the network on the receiving side. This is quantified by parameters including loss, delay, jitter, and cost. The parameters for

network service are determined by the network, but are affected directly by the communication layer's communication behavior. For example, an increased transmission rate over some interval may result in an increase in cost over that interval.

Channel Service: the behavior actually seen by the application. This will include quantities similar to network service, but measured at the interface between the communication layer and the application. The channel service is a function of communication configuration, communication behavior and network service.

In general these parameters will represent system behavior over some time interval, rather than instantaneous values. For purposes of this discussion, we consider them single-valued (i.e. points in some multidimensional space).

The channel's *service value* is a function of the channel service parameters. In order to solve the optimization problem of maximizing service value, the communication layer examines three relationships:

- The relationship between channel service and service value. We assume this is defined by a function provided by the application. This function defines the quantity to be maximized.
- The relationship between network service and communication behavior. In general, this can be defined by a curve (or plane) giving the value of the communication behavior that can be expected for each value of network service. We assume that the network service provides this curve.
- The relationship between communication configuration, communication behavior, network service and channel service, i.e. how the channel service delivered to the application is affected by the communication layer's protocol configuration, communication characteristics and the network service. For example, the use of retransmission as an error control strategy can decrease loss rate, but may increase delay. The use of forward error control, on the other hand, can decrease loss at the expense of increased bandwidth. (The assumption is that increasing delay or bandwidth results in decreased value.)

The characterization of this last relationship is a major challenge in the design and implementation of the communication layer. It might be realized through analysis, simulation, measurement, or (most likely) some combination of all three.

The remainder of this chapter describes an empirical approach to the implementation of such a communication layer. Section 2.2 discusses our approach for specifying the value of the channel service to the application. In Section 2.3, we address some of the issues involved in network service specification. And in Section 2.4, we present our techniques for using the information provided by these relationships to adaptively configure the communication.

2.2 Application Service Specification

Service requirements for distributed interactive applications vary for each user and across multiple users. We believe that it is important that knowledge about application-level service requirements be available when configuring the communication for the application. Toward this end, we support the user-level specification of the quantitative *value* of parameters of the communication service to the application. To clarify, consider a channel defined by the transmission of an image of interest to the user. Clearly, the quality of the image and, therefore, the resolution at which it must be

transmitted is directly related to the user’s current level of interest in the image’s contents. In this case, the actual “service value” ascribed to the channel may be computed with some application-specific “value function” that uses parameters quantifying the user’s current level of interest (e.g., distance from an image in a virtual world) and quantifying the actual level of service at which the image is transmitted by the network. The value of the service to the application may dynamically change, so that each value function represents the value to the application at a specific time or over a specific time period.

With a “value function”-based specification of desired service quality, it is possible to capture the tradeoffs the user is willing to make to realize certain levels of channel service. Typically, such tradeoffs represent relationships between conflicting service parameters. For example, in a lossy network, transmission rate may be sacrificed for high reliability data transfer. To study the utility of value functions in making such tradeoffs, we utilize particular value functions, called *payoff functions*. Payoff functions are associated with specific channels, and each function states the value to the application of receiving a certain level of service via this channel. Specifically, from the application’s point of view, the payoff function evaluated at a specific channel service represents the benefit gained by the application of achieving that specific channel service. Many such functions exist (often referred to as utility functions [63]) and may be formulated for the diverse multimedia applications considered in our work. In [34] and [22], such functions are formulated in the context of real-time application deadlines. [1] uses “reward” functions to capture similar application-specified service valuations, but limits these specification to “poor”, “good” and “excellent”. In comparison, our approach provides a more general interface for service valuation.

Figure 2 depicts two sample payoff functions addressing transmission reliability. The plot for Image 1 represents a payoff function for an image that has been requested at a range from 100-75% reliability. The shape of the curve within this range represents the relative value to the application of achieving that level of reliability. In this example, the application prefers 100% reliability and will still be somewhat satisfied at 90%, but will be less satisfied as the reliability approaches 75%, as expressed by the rapidly diminishing payoff values. Payoff is zero at the lower limit of acceptable reliability. In comparison, if the image can be accepted with reliability ranging from 40-80%, then the maximum payoff will be at the 80% reliability level, as shown in the plot for Image 2 in Figure 2. If the image is transmitted with more than 80% reliability, there is no additional payoff. Payoff diminishes until it reaches the limit of acceptable quality, in this case 40%. Payoff values are normalized to the range of 0 to 1 to permit comparisons between payoffs stated for different service parameters.

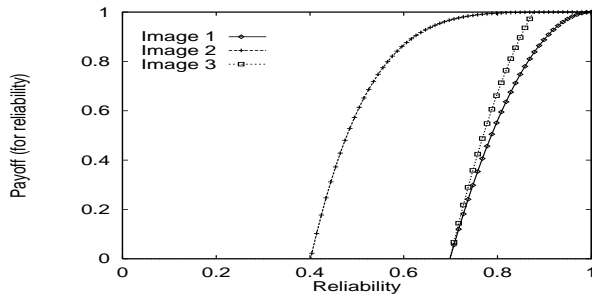


Figure 2: Sample Payoff Functions for Reliability

The specification of payoff for one service parameter (e.g., for reliability in Figure 2) does not capture the tradeoffs the application is willing to make between that parameter and others. To

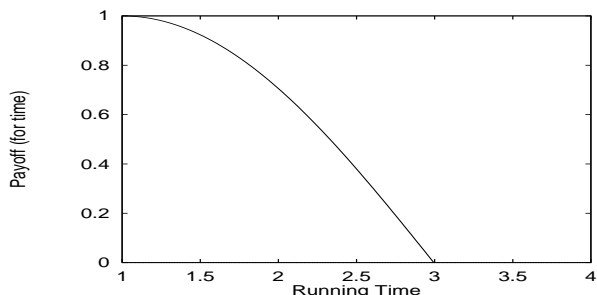


Figure 3: Sample Payoff Functions for Time

state such tradeoffs, the application must supply payoff information for a service parameter which represents some notion of “cost” to the application. For illustration, again consider transfer time vs. reliability. For example, in lossy networks, an increase in reliability will “cost” an increase in transfer time due to retransmissions. Figure 3 represents a payoff function for transfer time, where the application has requested a transmission rate from 1-3 time units, and has now specified payoff functions for two competing service parameters. Unrecovered losses will cause the payoff for reliability to decrease, but recovering those losses will increase the transfer time and decrease the payoff for transfer time. By supplying payoff functions for both parameters, the application provides the information necessary to determine how to find optimal operating points.

Payoff functions may also be used to represent the comparative value between data units. Consider two images of interest to the user, where the payoff functions for reliability for both of these images are those depicted by the plot for Image 1 in Figure 2. Such a formulation is adequate if the interest level of the user is the same for both images. In the case where one image is of less interest to the user, payoff for that image may be adjusted, as shown in the plot for Image 3. For this image, the payoff function expresses that maximum payoff is achieved for a lower reliability level as compared to Image 1, but the function still captures the shape of the original curve and also maintains the original minimum acceptable reliability level. Using these functions, the negative effects of transmitting Image 3 at a low reliability level are less than those experienced for Image 1.

The utility of payoff functions should be clear from the examples presented above: they may be used to impart to the providers of communication services application-level semantics with which differences in requested vs. delivered levels of service may be evaluated. In general, there is a payoff function, $P_d^p(l)$, for each service parameter, p , associated with each data, d , transferred at a service level, l . In this thesis, we focus on the service parameters *observed reliability* and *transfer time*. Given such service parameters and the payoff functions formulated in this section, we next discuss how to describe network resource availability appropriately so that payoff may be maximized for these services.

2.3 Communication Resource Specification

If the communication layer does not have knowledge of the available network service, it cannot adjust communication behavior appropriately. For example, a user may be receiving both data updates and continuous video. When the available network bandwidth is reduced, both communication streams will be negatively affected, one by dropping frames, the other by losing or delaying data. Such behavior is undesirable, especially if it can be remedied by lowering the quality of

the video stream in order to reduce bandwidth needs. Similarly, when increased network bandwidth becomes available, the communication layer should be able to respond by improving video quality. In general, then, opportunities for optimization are created by exposing network service information to effect adaptation of channel service.

The optimizations considered in our work explore the manner in which the communication layer can realize benefits for the application using knowledge about the relationship between requested vs. experienced channel service. For example, the communication layer may be supplied with a set of transmission rates and their respective loss rates, which jointly describe the current behavior of the available network service. In general, this implies that the communication layer has knowledge about what quality of service to expect for a range of different service requests. This relationship can be represented with *service availability curves*, as proposed in the form of Loss-Load Curves [68], which characterize the dynamic service a network can provide to its clients. Given a specific transmission rate for a sender, loss-load curves provide the sender with an expected loss rate. This information allows the communication layer itself to determine the tradeoff between higher output rates and higher loss rates. When service availability curves are not available, communication resource availability may be determined by the communication layer itself, perhaps by gathering statistics based on recent communication history [13] and by projecting such behavior into the near future [12].

The sample loss-load curve depicted in Figure 4 exhibits an increasing loss rate as the communication layer increases its transmission rate. In this example, the network is given a set of possible transmission rates and supplies the respective loss rates. A 10% loss is imposed for a transmission rate of 400KBps and increases 5% for each 100KBps increase in transmission rate.

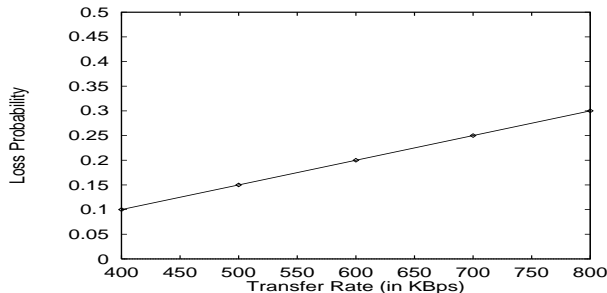


Figure 4: Example Loss-Load Curve

The specification of communication resources spans many parameters, including but not limited to delay, loss, jitter, available bandwidth and eventually some measure of monetary cost. The techniques described in this section provide a general approach to communication resource specification that is flexible and abstract enough to express current known and potential future communication service parameters.

2.4 Communication Adaptation

Payoff and service availability functions provide the communication framework with extensive information about an application’s service requirements and the network’s service availability. Consider the use of such information to configure communications during an application’s execution. Specifically, we employ *payoff adaptation* to optimize the use of available communication

resources. The payoff adaptation method employed in this work is reactive; it responds to changes in application requirements and network resources rather than attempting to anticipate them, as in predictive [11, 19] and learning-theory based adaptation methods [43, 36]. Payoff adaptation techniques allow us to balance application requirements like quality level and running time against resource availability. Resource information may be defined as execution time, available bandwidth, or message loss rate.

Two stages of adaptation. Payoff adaptation involves two stages. In the first stage, the communication layer determines a communication configuration for a specific point in time, given current application requirements and network service availability. In the second stage, the communication layer monitors changes in application requirements and network services. The effects of such changes on the service value are the determination of a new configuration.

1. *Using payoff functions to determine resource requirements.* This first stage of payoff adaptation uses the application-supplied payoff functions to evaluate application resource requirements. Given a fixed application resource specification and a fixed network resource specification, payoff adaptation considers a range of possible operating parameters and chooses those that maximize the service value to the application.

As an example, again consider the transfer of an image in the virtual environment. For this example, the application has specified two service parameters: ObservedReliability (OR), which captures the amount of the original data transmitted that is actually received, and TransferTime (TT), which captures the end-to-end transfer time for application data. To determine the maximum payoff, the communication layer executes the steps in Figure 5.

- (1) *for* ($i = \text{each ReliabilityLevel}$) *do*
- (2) *for* ($j = \text{each TransmissionRate}$) *do*
- (3) $\text{LossRate}_j = F_{\text{LLCurve}}(\text{TransmissionRate}_j)$
- (4) $\text{ObservedReliability}_{i,j} = F_{or}(\text{ReliabilityLevel}_i, \text{LossRate}_j)$
- (5) $\text{TransferTime}_{i,j} = F_{tt}(\text{ReliabilityLevel}_i, \text{LossRate}_j, \text{TransmissionRate}_j)$
- (6) $\text{TotalPayoff}_{i,j} = Q(P^{or}(\text{ObservedReliability}_{i,j}), P^{tt}(\text{TransferTime}_{i,j}))$

Figure 5: Optimization Algorithm

To evaluate individual service parameters, we need information about the relationships between multiple communication variables as discussed in Section 2.1. This relationship may be obtained via a function which maps communication configuration, communication behavior and network service into channel service. In general, such interrelationships may be collected using profiling techniques prior to running the application, from an analytical model, or they may be gathered by the application during execution (e.g., based on recent history). In the context of our example, the information supplied by the loss-load curve in Figure 4 provides the communication layer with a mapping from transmission rate to loss rate, as shown in step (3) above. In Figure 5, the functions F_{or} (step (4)) and F_{tt} (step (5)) supply the mapping from reliability level (communication configuration), transmission rate (communication behavior) and loss rate (network service) to the channel service parameters. These functions calculate $OR_{i,j}$ and $TT_{i,j}$, representing the service parameters evaluated at the specific reliability level, i , transmission rate, j , and loss rate.

In the following step, the communication layer evaluates the benefit to the application of operating with these service parameters. As discussed in Section 2.2, there is a payoff function, $P_d^p(l)$, for each service parameter associated with a data and service level pair. Since the application indicates multiple service parameters, the payoff for each of these parameters must be considered

to determine a net payoff (step (6)). In other words, the net payoff, P_{total} , is a function, Q , over all the defined $P_d^p(l)$, where p represents the service parameter, l represents the service level and d represents the specific data in question. In the case of an image with two payoff functions, one for reliability and one for transfer time, the payoff for the transfer of the image is determined by evaluating the payoff functions, $P^{\text{or}}(\text{OR}_{i,j})$ and $P^{\text{tt}}(\text{TT}_{i,j})$, at a specific reliability level, i , and transmission rate, j . The function Q , which defines the composition of payoffs for multiple application parameters, is supplied by the application. The application may also use a default composition function, such as multiplication. Multiplication has the advantages of retaining payoff values between 0 and 1, as well as causing total payoff to go to 0 if any individual payoff is 0.

These steps are repeated to determine the payoff for each possible operating point (as indicated by the loops in steps (1) and (2)). From these operating points, the communication layer chooses the reliability level and transmission rate that maximizes total payoff for the application.

Our approach to choosing an operating point given payoff and resource availability functions has the following advantages:

- It is extremely flexible because it makes no assumptions about the form of the payoff and resource availability functions, treating them as "black boxes". The functions might be analytical formulas, graphs (sets of ordered pairs), or even programs in a suitable language.
- The cost of computing the operating point is simply determined by the cost of evaluating all of the functions at a single point, multiplied by the number of potential operating points considered. Thus, it should be relatively easy to predict the cost of determining an operating point.
- There is an obvious tradeoff between the optimality of the chosen operating point and the granularity of approximation, i.e. the number of operating points considered.

Given payoff and/or resource functions that have certain characteristics (e.g. continuous and monotonic), it may be possible to eliminate potential operating points from consideration.

2. Adapting to parameter changes. The second stage of payoff adaptation involves adapting to changes in both application requirements and network resources. As application requirements change, the application may provide the communication layer with new payoff curves for its data, or it may simply change the characteristics of its communication. At this point, the communication layer must reevaluate the communication configuration and determine the need for reconfiguration. Reconfiguration decisions may be caused by changes in network resource parameters, which may be provided by the network in the form of a new loss-load curve, or derived from observing the network.

The benefits of using payoff-based adaptation are derived from the richness of the information provided to the communication layer. The communication layer is given a wide range of resource allocations that will satisfy the application and has the information necessary to determine what inside this range would best benefit the application. This freedom allows the communication layer to base adaptations on the requirements of the application. In comparison, adaptation schemes that use specification ranges or regions have the freedom to work within those areas, but have no guidance as to what would benefit the application. We can emulate such functionality by setting the service levels within the range or region to 1 and the payoff for the area outside the range to 0.

Chapter 3

Dynamic Communication Configuration Framework

An important tool for the adaptive resource management techniques described in Chapter 2 is the ability to reconfigure communication protocols and communication protocol parameters during active data transfer. Unfortunately, current network protocol technology provides very limited control over such communication resources. In an effort to expand the basic service provided by standard protocol stacks, a number of systems provide a choice of different or even customized protocol stacks, but do not provide any support for dealing with dynamic variations in ongoing communications [29, 32, 52, 9]. In comparison, our research provides customized protocol stacks with an easily accessible model for communication protocol configuration. Based on this model, communication protocols may be managed for the applications that use them. This model builds on protocols that are configurable at runtime and aims at applications that have been constructed to utilize such configurability.

3.1 Example Application

Many multimedia applications share the characteristic that communication requirements differ across the multiple data streams of the application, (e.g., voice vs. video), as well as across the frames of an individual stream (e.g., I-frames vs. P-frames vs. B-frames in MPEG). It is also a common characteristic that applications transmit secure and non-secure data within the same data stream. Such highly variable application data characteristics are the target of this research, as evidenced by our multimedia slideshow.

Specifically, in this application, any number of users are given the ability to transmit continuous voice, sound files, image files, and text to each other. Users may initiate a talk session, establish any number of connections for data transfer, stop watching a show, etc. In addition, instead of creating separate connections for each data stream being transmitted, users specify virtual channels for each data stream and the protocol framework handles how these channels are distributed across the available connections. Toward this end, a generic connection interface (see Figure 6) is provided by our configuration protocol framework.

In this context, applications may have multiple data streams, each of which utilizes a separate communication channel. The communication framework then maps the communication channels to one or more actual network connections. Figure 6 presents an example of three data streams, each mapped to one communication channel. The three channels are then multiplexed onto one network connection. It is also possible that a data stream may be split into multiple channels. For example, an MPEG transmission may use three channels, one for each frame type. Additionally,

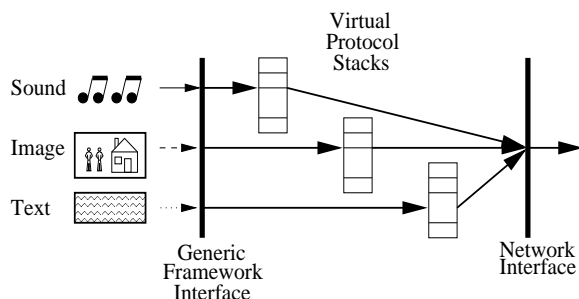


Figure 6: Multimedia Slideshow

application data may be split across multiple network connections based on the needs of the data stream and the characteristics of the connection.

Application data is then transferred via these virtual channels that may employ multiple and differing protocol stacks, selected on a per stream or per message basis. For example, application data may initially be transmitted in the form of unencrypted text. At some point during execution, security may be increased simply by providing an additional parameter to the communication framework that selects a more secure protocol stack, either by adding a security protocol to the existing channel or by creating a new channel with the appropriate protocols. Similarly, for certain critical communications, highly reliable transport protocols may be employed while other, less important communications may be sent less reliably. For example, in the case of slide shows, an “itinerary” slide may be sent reliably, while multiple slides depicting the actual images may be sent less reliably. In any of these examples, the framework to makes use of a new protocol configuration with the appropriate protocols, instead of being forced to remove an existing connection, establish a new connection, then start using it.

3.2 Framework Design

The framework is designed around three major components. The configurable protocol subsystem provides the ability to dynamically reconfigure communication protocols without needing to tear down and rebuild connections. The interface to the application enables the specification of application resource requirements in the form of multiple payoff functions plus information on how these functions should be combined. The interface to the network enables the specification of service availability information or monitoring information from the network. All three of these components provide dynamic service.

3.3 Background and Related Work

This section briefly relates our work to other research in dynamic protocol configuration. The protocol stack is a commonly used abstraction for the protocol processing associated with a certain communication channel. This stack defines the nature and order of protocol processing and in general, each such stack utilizes a number of protocol modules to provide a specific communication service. *Configurable communication protocols* permit applications to construct their own protocol stacks in order to provide appropriate services with the desired functionality and performance. In

practice, an application may construct and use multiple such stacks during its execution.

Existing configurable protocol systems may be distinguished on the basis of when configuration is performed:

Connection time configuration – by configuring its communication at connection time, applications may employ multiple communication channels with differing service requirements. However, once a communication channel has been configured, it cannot be changed. As a result, any changes in service requirements experienced by the application requires that the connection be torn down and re-established with different requirements. In addition, such a connection is limited to handling some single type of data chosen at the time of channel establishment. [29, 32, 52, 9]

Dynamic configuration – by permitting applications to configure a communication channel during its operation, changing service requirements may be accommodated, and changes may be made concerning the types of data communicated via that channel. Such dynamic configuration is implemented by the subsystem described in this section. [61, 55, 71]

A number of proposals have been made for providing dynamic communications through configurable protocol systems. The goal of ADAPTIVE (*A Dynamically Assembled Protocol Transformation, Integration, and eValuation Environment*) [61] is to provide automated support for composing lightweight and adaptive protocols. This approach employs a collection of reusable “building block” protocol mechanisms that may be composed automatically at runtime. This work emphasized the need for dynamically configurable protocols. Da CaPo (**D**ynamic **C**onfiguration of **P**rotocols) [55] is another approach to modular configurable protocols. In Da Capo, configuration is done with respect to application requirements, properties of the offered network services and available resources in the end system. This work differs from ours in that the connection manager, configuration manager and resource manager are built into the framework. Although these are important components of a dynamically configurable protocol systems, our implementation separates them out from the main functionality of providing configurable communication. Zitterbart, Stiller and Tantawy [71] also describe a communication subsystem that allows applications to request individually tailored services. We provide results that demonstrate the necessity for this type of refinement.

3.4 The Configurable Protocol Subsystem

The basic design ideas for the configurable protocol subsystem have been explored in [15, 14]. The Tau system described there is intended for composition of end-to-end protocol functions. It has been partly implemented for purposes of our research. Specifically, the metaheader protocol originally described in [15] includes the application-level concept of *protocol configuration* and *configuration attributes*. A configuration represents a certain protocol stack associated with a connection, and an attribute is a parameter of that configuration exported to and manipulable by an application. Besides protocol configurations, the three additional relevant components of the protocol subsystem are:

- The **protocol infrastructure** provides the ability to maintain state information for dynamically constructed protocol configurations.
- The **metaheader protocol** provides the information necessary to correctly process each message.

- The **protocol functions** provide the specific functionality for independently processing application data.

The remainder of this section describes the design components of the configurable protocol subsystem: protocol configurations, the protocol infrastructure, the metaheader protocol and the protocol functions.

3.4.1 Configurations

A *configuration* is a labeled set of protocol modules. In essence, a configuration represents a virtual protocol stack used within a communication channel. During data transmission, the configuration defines the processing order for the protocol stack. The original design of the Tau system allowed for parallel processing of protocol functions. The configurable protocol subsystem implemented for this research only allows sequential protocol processing. This was an implementation decision and does not limit the flexibility of our system.

Any number of configurations may be defined for each channel. An application determines which configuration is used for which data. For example, the application may specify that data may be transmitted with and without a security protocol. The channels would be setup for both configurations and the application would specify which data should be sent with and without the security protocol.

For a specific configuration, two levels of changes may be enacted. The first level affects the processing of messages by a certain configuration, performed by manipulation of *configuration attributes*. When a configuration attribute is changed, all messages using this configuration are affected. For example, consider a configuration that includes JPEG compression. Internally, JPEG uses some parameters that affect compression processing, some of which may be exported as attributes to the application level. If such attributes are manipulated to change from fast to slow JPEG, for instance, then all future communications via this configuration would be affected. However, any other configuration also using JPEG is not affected, since each configuration maintains its own state.

The second level of configuration changes concerns switching configurations (i.e., protocol stacks) for existing channels. For instance, consider the JPEG example used above, where the existing configuration sends messages using both JPEG and DES. For a certain message being sent, a change from DES to IDEA encryption, for instance, would entail (1) creating a new configuration that contains both JPEG and IDEA, if not already present, and (2) switching to this new configuration for future messages being sent. Another example of such dynamic protocol configuration is one in which selected messages (e.g., those containing some control information) have to be processed with some control protocol and a different encryption key. In this case, most messages containing JPEG image data would continue to be sent with the protocol specified by the original configuration. In addition, a new configuration would be created that includes the control protocol and DES. Control messages would be directed to this new configuration. In this example, image and control processing would continue in parallel, both using the same channel but using different configurations.

3.4.2 Protocol Infrastructure

The primary design objective of the *protocol infrastructure* is to provide two mechanisms: protocol function composition and multiplexing. This functionality is provided in a manner that supports various performance-enhancing techniques, while preserving modularity in some form. The idea is

that these mechanisms should work with an extensible set of policies, in order to support a wide variety of applications, including those whose requirements are not yet fully understood. Our goal is a generic model of protocol processing, in which the protocol functions are separated from the details of the “glue” that binds them together.

Because protocol functions are not layered in the protocol infrastructure, they do not attach their headers directly to outgoing data units, nor extract them from incoming data units; instead, header placement is handled by the protocol infrastructure. The generic protocol model defines the interface between the protocol infrastructure and each protocol function. The requirement that this interface be specified represents an opportunity to reduce the costs of porting protocol implementations by isolating, as far as possible, the specification of a protocol’s function from the “glue” used to combine it with other protocols. This is a key design motivation of the protocol infrastructure.

A logical block diagram of an implementation is shown in Figure 7. Each protocol function module is viewed as a (passive) transducer, which is given inputs (state information, control parameters, incoming header, and possibly data) and produces outputs (updated state information, control parameters, outgoing header, and processed data). The architectural “glue” is provided by the demux-and-dispatch function. It selects and coordinates between the protocol functions, providing them with inputs based upon external events, and collecting and passing on to the external environment (i.e. the user, the network, auxiliary functions such as timeout and buffer management) their outputs.

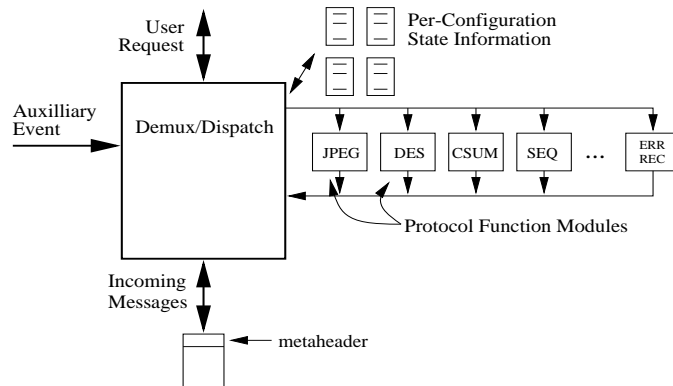


Figure 7: Architecture of Protocol Subsystem

3.4.3 Metaheader Protocol

The *metaheader protocol* provides the information necessary to process each message by specifying the protocols involved in each communication and by implementing the multiplexing and composition mechanisms. A *metaheader* is an extended message header that provides enough information for that message to be handled and processed correctly at the receiving end.

The metaheader is built from three building blocks: message header information, protocol header descriptors and individual protocol headers (see Figure 8). The message header information provides the information necessary to understand how the rest of the metaheader was built. It includes header length, number of protocol headers, sender and receiver application identifiers, and sender and receiver configuration identifiers. The protocol header descriptors and the individual

protocol headers come in pairs. The header descriptor determines which protocol is next and defines the length of the specific protocol header. The individual protocol headers are defined by the specific protocol. Combined, these three building blocks provide the receiving side with a map of how the message needs to be processed at the receiving side.

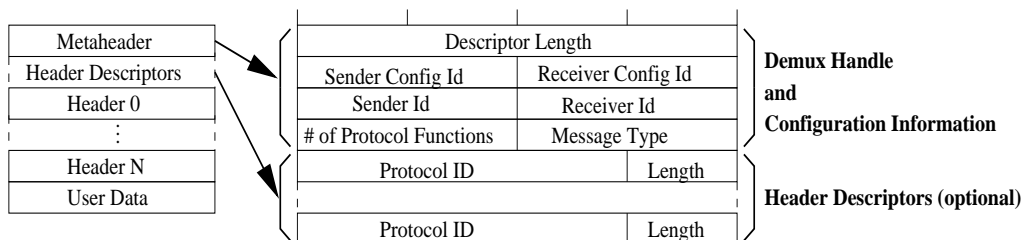


Figure 8: Example Message using Metaheader Protocol

3.4.4 Protocol Functions

Each *protocol function* defines a specific function that can be processed independently of other protocol functions. This functionality can vary in complexity from simple checksumming to the complete functionality of the TCP protocol. Such protocol functions provide the “menu” from which applications can choose the services they desire. We envision communication services implemented by composing atomic single-function protocols from a “menu of functionality”, as have others [52, 71, 29, 55, 61]. For example, a service for a reliable, secure image application could be implemented with JPEG, DES, a sequence numbering function, and two different reliability functions (one for request retransmission and one for response error detection and retransmission).

As an example, Appendix D lists the pseudo-code for the variable reliability protocol (described in Chapter 5) that was implemented as a protocol function in our configurable protocol subsystem.

3.5 Interfaces and Implementation

The design of our subsystem distinguishes four interfaces (see Figure 9). First, the subsystem provides applications with an interface used for direct interaction with the subsystem. These functions include all entry points necessary for configuration creation and for sending and receiving data. Second, applications can retrieve information about a specific protocol in a specific configuration, using configuration and protocol module identifiers, and attributes. Third, in order to have a generic interface between the subsystem and protocol functions, each protocol provides a set of standard entry points that are accessible by the subsystem. Fourth, there is a standard interface for communication with the network. The resulting on-line control based on network or application feedback may be performed by the application or by a separate control entity (see Figure 9).

3.5.1 Application to Protocol Infrastructure Interface

The key idea behind protocol configurability is that application should be allowed to choose what protocols are needed for a specific communication channel, stream, or even message. The runtime realization of such choices is the *configuration*. Once the appropriate configurations are established for each connection, applications can use a *configuration handle* to tell the communication layer

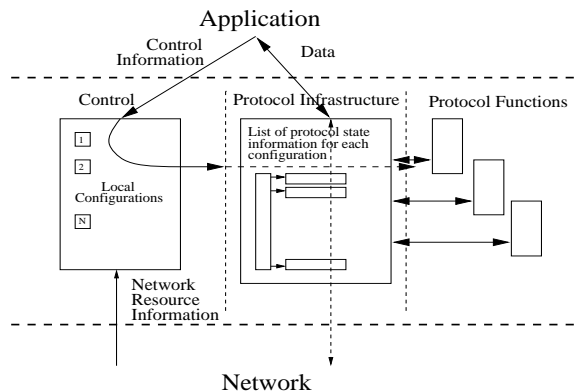


Figure 9: Interfaces between Application, Protocol Infrastructure, Protocol Functions, and Network

which configuration is to be used for each message. The processing order of the protocol functions in a configuration is determined by the order in which functions are added to the configuration. Outgoing messages are processed in order; incoming messages are processed in reverse order. In order to create a configuration, applications follow these steps:

1. Create the configuration :

```
ConfigId createConfiguration()
```

This function returns a handle to a new, empty configuration. This handle is used by applications to specify this configuration in the future.

2. Add protocols to the configuration:

```
addProtocol(ConfigId, ProtocolId, ProtocolState *)
```

This function adds a protocol to the configuration, allocates memory for the protocol's state and header information, and initializes the protocol. This generic interface gives the framework the ability to set up any necessary information for a protocol without having to know the specifics of that protocol.

3. Add a transport protocol:

```
addTransportProtocol(ConfigId, ProtocolId, ProtocolState *)
```

This function specifies the transport protocol (i.e TCP or UDP) for this configuration. This particular distinction is necessary in our current implementation in order to operate in user space on top the standard implementations of TCP and UDP.

The generic send/receive functions provided by the framework permit the sending and receiving of data in conjunction with a specific configuration:

- **sendData (Data *, Length, ConfigId, Label *)** – “ConfigId” identifies the configuration to be used for the processing of each message. “Label” provides applications with the ability to send control information out-of-band. The framework takes this label and includes it in the message similarly to a protocol header. Providing this label allows applications to support the concept of application level framing (ALF) [18].

- `recvData (ConfigId *, Data *, Length *, Label *)` – since applications do not have foreknowledge of what type of messages they might be receiving, or which configuration id might be used, the framework provides this information. As new configurations are used on the sending side, new configurations are dynamically built on the receiving side. In general, applications may or may not need to know what configuration was used to process an incoming message. By knowing the configuration id, applications can access application-specific state information about this message, if necessary.

Appendix A lists the pseudo-code for the sending and receiving functions of the configurable protocol subsystem .

3.5.2 Application to Protocol Functions Interface

A set of special purpose functions are provided to give applications the ability to communicate directly with existing protocols. These functions allow applications to set and retrieve application-specific protocol parameters, called *attributes*:

- `resetProtocol (ConfigId, ProtocolId, ProtocolState *)` – set up the appropriate attribute values. This function permits the setting of specific protocol attributes. The protocol takes this information and makes any internal changes it deems necessary.
- `getProtocolState (ConfigId, ProtocolId, ProtocolState)` – inspect the values of specific attributes.

These functions are used when applications need to inform a protocol of attribute changes, or when applications need information about the state of a protocol. If we consider the example of changing the JPEG compression parameter described in Section 3.4.1, an application would use the function `resetProtocol`. In a situation where a flow control protocol is being used, an application may want to query the flow control protocol to see if the current message can be transmitted. In this case, the application would use the function `getProtocolState`.

3.5.3 Protocol Infrastructure to Protocol Functions Interface

This interface is defined in terms of a set of entry points corresponding to various events. Currently, each protocol must provide the entry points:

- initialization;
- send processing;
- receive processing;
- acknowledgment processing;
- timeout processing; and
- resetting or changing protocol attributes.

When control is passed to a protocol function, it receives a set of parameters, which may include some or all of the following:

- local control information (e.g., data size, user parameters, destination application identifiers);

- remote control information (the header associated with an incoming message);
- user data; and
- persistent state information relevant to the connection or endpoint.

3.6 Runtime Configuration

The goal of the dynamic communication configuration framework is to provide the mechanisms necessary to control application communications. The configurable protocol subsystem described above supplies the pieces necessary to make these changes. Our framework also contains the engine for determining when these changes need to be made and what the changes should be. The decisions are based on information supplied by applications through the interface between the application and the framework. In this way, applications supply the policies with which the mechanisms are used. This interface defines the type of information used. In our framework, this interface allows a detailed description of application resource requirements in the form of payoff functions. Applications also supply the function with which these payoff functions are combined. By enhancing the interface to include such detailed information, the framework is given a well defined space from which it can make well informed decisions.

When determining when and how to reconfigure communications, the framework also considers network resource information. The interface between network services and our framework allows the network service to define a mapping between service availability and requested service. By including this information, the framework is provided more detailed information about the space from which it makes its configuration choices.

Experimental results presented in Chapter 6 highlight the connection between configurable protocols and adaptable applications. Our experience is consistent with the theory that given the ability to configure application protocols during runtime, applications can simplify their design and improve their performance.

3.7 Overview of Contributions

The research contributions of this work are the design and implementation of a framework for configurable communication protocols providing application-level access to protocol stacks for purposes of runtime configuration. This dynamic communication configuration framework provides the core functionality of our cooperative communication layer. By utilizing the framework's runtime configuration of the communication protocols used during data transfer, the cooperative communication protocols is able to realize tradeoffs across multiple service parameters.

The framework has been implemented on multiple platforms (SunOS, Solaris, IRIX, Linux). The configurable protocol subsystem has been implemented and is used as a communication base for all experiments presented in this thesis. Communication protocols implemented in our subsystem include image compression (JPEG), audio compression (GSM, ADPCM), data compression (LZO), data encryption (DES,IDEA,MD5), leaky bucket rate control, power management (described in Chapter 4), variable reliability (described in Chapter 5), and selective acknowledgment based data transmission (described in Chapter 7). The functionality for controlling reconfiguration based on payoff functions has been implemented.

Chapter 4

Mobile Communication and Power Management

The first step towards dynamic configuration is understanding what resources are important to an application. Only with this understanding can we work towards providing better service for the application. In the context of mobile computing, applications focus on two resources: power and transmission quality. In this chapter, we present a power management protocol which allows the application to conserve battery power during idle times in communication. The gain to the application is measured in extended battery lifetime. The cost to the application is an increased delay for unexpected data from the base station. The determination for when to use power management is based on the communication patterns of the application. For this application, the goal is to balance power management with limiting the delay imposed on incoming data.

4.1 Power Management for Mobile Hosts

In today's world of mobile communications, one of the most precious commodities is power. The mobile host can only operate as long as its battery maintains power. New machines are being made to use less energy allowing for smaller batteries with smaller capacities. The trend in mobile computing is towards more communication-dependent activities, with mobile users switching from traditional wired Ethernet communication to wireless communication (using wireless Ethernet cards, for example). When inserted, many wireless communication devices consume energy continuously. Although dependent on the specific machine and wireless device, this energy consumption can represent over 50% of total system energy for current hand-held computers and up to 10% for high-end laptops. These trends make it imperative that we design energy-efficient communication subsystems.

Various techniques, both hardware and software, have been proposed to reduce a mobile host's energy consumption during operation. Most software-level techniques have concentrated on non-communication components of the mobile host, such as displays, disks and CPUs. In particular, researchers have looked at methods to turn off the display after some period of inactivity (as often implemented in BIOS or screen savers), to spin down the hard disk of the mobile host [25, 30, 44], and to slow down or stop the CPU depending on work load [28, 45, 67]. The principle underlying the techniques for controlling these components is to estimate (or guess) when the device will not be used and suspend it for those intervals. Stemm et al [65] have identified the problem of excess energy consumption by network interfaces in hand held devices, and have provided trace-driven simulation results for simple software-level time-out strategies. The new IEEE 802.11 standard that is being adopted by some vendors adopts lower level solutions (at the MAC and PHY layer) to support idle-time power management. Hardware-level solutions for managing the communication device focus on modulating the energy used by the mobile transmitter during active communication [49, 57, 60].

Our research presented in this chapter focuses on software-level techniques for managing the mobile host's communication device through suspension of the device during idle periods in the communication. We present a novel transport level protocol for managing the suspend/resume cycle of the mobile host's communication device in an effort to reduce energy consumption. The management of communication devices creates a new and interesting challenge not present when managing other devices' energy consumption. Similar to hard disks and CPUs, the communication devices continuously draw power unless they can be suspended. A suspended hard disk or CPU can be restarted by any user requiring that device. However, when a communication device is suspended, the mobile host is effectively cut off from the rest of the network. A mobile host with a suspended communication device can only guess about when other hosts may have data destined for it. If the suspension of the mobile host's communication does not match prevailing communication patterns, the isolation can cause buffers to overflow both in the mobile host and in other hosts trying to communicate with it. Additionally, other hosts may waste precious resources trying to communicate with the mobile host if they have no knowledge about whether or not the mobile host's communication is suspended.

Our goal is to provide mechanisms for managing and reducing the energy consumption of the communication device. We present a simple model for mobile communication that provides adaptable functionality at the transport layer for suspending and resuming communication. By exposing this functionality to the communication layer, we enable application-specific solutions to power management. The communication layer utilizes information about the service requirements of the application in the form of payoff functions for service quality and energy consumption. Additionally, the communication layer uses energy consumption models based on the communication patterns of the application and the service provided by the network. Energy savings are attained by suspending communications and the communication device for short periods of time. During these suspensions, data transmissions are queued up in both the mobile host and any other host trying to communicate with the mobile host. The key to balancing energy savings and data delay lies in identifying when to suspend and restart communications. By abstracting power management to a higher level, we can exploit application-specific information about how to balance energy savings and data delay.

Intuitively, energy conservation is achieved by accumulating the energy savings from many small idle periods. We, however, need to be careful to monitor any additional energy consumption caused while executing the suspend/resume strategies. Additionally, we need to consider the effect on other hosts who are trying to communicate with the suspended mobile host. A base station using our protocol has enough knowledge about the state of the mobile host to know when it is suspended and can use this information to help employ scheduling techniques. We implemented our protocol and experimentally determined its effect on energy consumption and the quality of communication. Using three simulated users designed to capture typical mobile communication patterns and three trace-based web users, we obtained 48-83% savings in the energy consumed by the communication subsystem, while introducing a small additional response delay (0.4-3.1 seconds depending on the power management level) that is acceptable for many applications, like web browsing.

In Section 4.2, we present our basic mobile communication model, and the important issues in power management for communication. In Section 4.3 we present our power management protocol and discuss the effect of timing issues on the effectiveness of our protocol. Section 4.4 describes our experimental setup and the communication patterns used in our experiments. We then present measurements from the implementation of our power management protocol and discuss the results in the context of several real systems. In Section 4.5, we discuss adaptive control strategies and we present our conclusions in Section 4.6.

4.2 Communication Model and Power Management

The introduction of wireless links into communication systems based on wired links has posed a number of problems. These problems include different loss characteristics and different bandwidth capabilities on the wired and the wireless line, synchronization of disconnected operations, and issues involving packet forwarding. These problems pose significant challenges for end-to-end communication protocols. Two types of models have been studied [5]. The first model exploits the natural hop between a base station and the mobile host for communications crossing the wired-wireless boundary. Standard communication protocols are used by wired hosts to a base station and specialized protocols are used for the final hop from the base station to the mobile hosts [4]. The second model utilizes and tunes existing end-to-end protocols, providing help and hints along the way [6].

In this chapter, we focus on the first model of communication described above, which allows us to isolate the communication between the base station and the mobile host. Although we do not address the second model in this thesis, with some extensions, this technique is also applicable to the second model. We target our approach at the transport layer, where we provide a set of mechanisms that allow communication to be suspended and resumed. We assume a model where the mobile host is communicating with the rest of the network through a base station. This base station may be a proxy, or it may be the connection point for end-to-end communication with other hosts. Often, dealing with mobility does not fit into the standard seven layer model. By exposing power management techniques to the application, we provide a system-level solution aimed at end-to-end communication. For our experiments in this chapter, we concentrate on the communication between the mobile host and the base station, and for clarity assume that all communication to and from the mobile host is directed through one specific base station. This work can be extended to include changing base stations through techniques similar to those used in [4, 6].

Current wireless communication devices typically operate in two modes: transmit mode and receive mode. The transmit mode is used during data transmission. The receive mode is the default mode for both receiving data and listening for incoming data. Much of the time, the wireless communication device sits idle in receive mode, and, while the power required for reception is less than the power required for transmission, the energy consumption is not negligible. A number of solutions aimed at power management at the MAC or PHY layer have been proposed. With the IEEE 802.11 standard, compliant cards can exchange information about outstanding data to decide on when to wake up suspended cards. There are ongoing efforts to provide IEEE 802.11 compliant support for power management by introducing new features into the next generation wireless communication cards [35]. A comparison of additional MAC layer protocol solutions can be found in [16]. Such approaches that rely solely on techniques provided by the device cannot take application specific information into consideration when determining power management strategies. In comparison, our approach exposes power management efforts to the application, allowing for better informed decisions as how much and when to use power management techniques. Researchers have also considered hardware-level solutions to provide low power communication capabilities [49, 57, 60]. Such solutions reduce the power cost of operating in either one of the modes, and are orthogonal to our approach which addresses the amount of time the device spends in each mode.

Logical areas to look for software-level power conservation in communication are two-fold. Since data transmission is expensive, we can reduce the time spent in transmission. This can be achieved by data reduction techniques and intelligent data transfer protocols. The obvious technique of data compression reduces the amount of transmission time, but requires additional CPU cycles for performing compression. The connection between compression and communication

rates is studied in [23]. Through simple experiments, we observed that, considering the current power requirements of CPUs versus wireless communication devices, the benefit in terms of energy savings from reduced communication time often outweighs the increased energy consumption costs for compression. Intelligent data transfer protocols can be used to reduce the effect of noisy connections that cause power-expensive retransmission of lost messages. Our continuing research addresses the assessment of the effects of different techniques for data reduction, including reduced reliability requirements, and their effect on both power reduction and communication quality. In Chapter 7, we address the issue of transmission time power management.

The second area, and the emphasis of this chapter, is the cost of leaving the communication device sitting idle during periods of no communication activity. During such idle periods, the communication device draws power listening for incoming data. Our goal in this work is to reduce the amount of time the device sits idle drawing power by judiciously suspending it. Suspending a wireless communication device is similar to slowing a CPU in that there are some small power costs associated with suspension and resumption. As mentioned in Section 4.1, the difficult part here is to deal with when to suspend and resume the communication device, how to deal with the mobile host being unreachable at times, and how to address the issue of not losing en-route data. Our protocol and its implementation presented here address these problems. Since the protocol itself generates additional communication during these idle periods, there needs to be a balance between when it is beneficial to use the power management techniques, and when we should leave the device on continuously.

In contrast to the solutions proposed by the IEEE 802.11 standard, we believe that power management should be controlled by the mobile host, potentially even the application. By providing power control at the transport layer (or above), we can provide power management interfaces to the application, allowing the application to better control the communication, enabling adaptive power management driven by the needs of the application. Specifically, communications using the IEEE 802.11 standard will always pay the overhead of delays imposed by using power management, while our techniques allow the application to determine when such delays are too high, and so adapt power management levels. Stemm et. al [65] have also investigated methods for reducing energy consumption of network interfaces, specifically targeting their research at hand-held devices. Their research suggests application-specific solutions to such problems. In contrast, our research provides a general solution capable of hosting various strategies, both static and adaptive. Our measurements are with a real implementation of a power management protocol in an experimental setup. We are, therefore, able to observe the effects of the queuing of data and the real effect of extra energy consumption by such a protocol. We measure the energy consumption in the context of the entire system, considering such costs as message processing and disk accesses, for various simulated workloads that we expect mobile users to perform.

4.3 Communication-Based Power Management

Currently, a typical mobile host leaves its wireless Ethernet card in receive mode during the time it is not being used, unless the user explicitly removes the card. The technique described in this section provides mechanisms to extend battery lifetime by suspending the wireless Ethernet card during idle periods in communication. At the heart of the technique lies a protocol where the mobile host acts as a master and tells the base station when data transmission can occur. When the mobile host wakes up, it sends a query to the base station to see if the base station has any data to send. This permits communication device suspension at the mobile host, and enables the implementation of communication scheduling techniques at the base station. The suspend/resume

cycle results in bursts of communication that may be followed by periods of inactivity. Although producing such bursty communication may incur additional delay, bursty communication patterns lend themselves well to efficient scheduling techniques.

With the suspension of a communication device, a mobile host will experience an additional delay in data transmission since data on both the sending and receiving sides may be held up during suspension. The mobile host can monitor its own outgoing communication patterns to insure that, despite these suspension times, communication continues smoothly without buffer overflow. The base station, on the other hand, has no means to restart communication if it notices that it is running out of buffer space. It is up to the mobile host to understand the base station's expected communication patterns so that the buffers at the base station do not overflow. In order to efficiently use our power management techniques, our communication layer must monitor the communication patterns of the mobile host and match the suspend/resume cycle to these patterns.

The protocol we describe in this section allows a mobile host to suspend a wireless communication device. Periodically, or by request from the application, the protocol wakes up and reinitiates communication with the base station. In the rest of this section, we will describe our power management protocol in detail and discuss the significance of some of the timing parameters. Appendix B describes in detail the commands used by the protocol and the possible states and state transitions for both the master and slave.

4.3.1 Power Management Control Protocol

The power management control protocol provides control of the transmission of data between a specific mobile host and the base station with which it is communicating. The protocol is implemented in user space as a protocol in our dynamic communication configuration framework. The protocol makes no assumption about the underlying transmission protocol being used for data transfer (i.e., TCP or UDP), but assumes reliable transfer of its control messages.

In this protocol, the mobile host is the master and the base station acts like a slave. The slave is only allowed to send data to the master during specific phases of the protocol. During non-transmit phases, the slave queues up data and waits for commands from the master. Idle periods for both the master and the slave can be detected through the use of idle timers or indicated to the protocol from the application. In the protocol state diagrams for the master and the slave (Figure 10 and Figure 11), **IN:** indicates an input event that can be either an incoming message or a timeout, **Q:** indicates the state of the queue, and **OUT:** indicates an outgoing response message.

As shown in Figure 10, the slave is initialized to be in the **SLEEPING** mode. It can only leave that mode upon a **WAKE_UP** message from the master. If the slave has data to send, it will enter the **SEND_RECV** mode. The slave will stay in this mode until it has detected that it has no more data to transmit, whereupon, it will send a **DONE** message to the master, enter the **RECEIVING** mode, and continue receiving until it receives a **SLEEP** message. If during this time the slave detects that there is new data to transmit, it will send a **NEW_DATA** message to the master and enter the **RECEIVING_WAIT** mode. The slave can only start to transmit when it receives a **WAKE_UP** message from the master. If a **SLEEP** message is received first, the waiting data stays buffered and is not transmitted until the next resume cycle.

Although the state diagram for the master (Figure 11) is much more complex, we can see that the states may be partitioned into three sets. The first set (**SLEEPING**) concerns the master when it is sleeping. When the master is in the **SLEEPING** mode, it can be woken up by one of two triggers: a wakeup timer or new data to transmit. If the wakeup timer expires, the master sends a **WAKE_UP** message along with any new data to the slave. If there is new data to transmit to the slave before the wakeup timer expires, the master has the option to wake up and transmit this new data, or

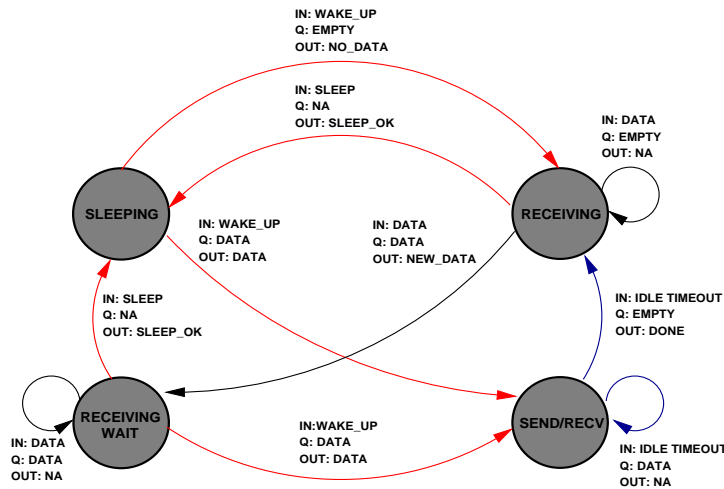


Figure 10: Slave (Base Station) Protocol State Diagram

continue sleeping and queue up the data until the timeout expires.

The second set of states (**SENDING_WAIT**, **WAITING**, and **WAIT_FOR_OK**) concerns the master when it is waiting for a response from the slave about whether or not the slave has data to send. In the **SENDING_WAIT** mode, the master is transmitting data and in the **WAITING** mode it has no data to transmit. When the master receives a response from the slave in the form of a **DATA** or a **NO_DATA** message, the master enters the appropriate state in the third set. Additionally, if while in the **SENDING_WAIT** mode an idle timer expires indicating that the master has no more data to send, the master enters the **WAITING** mode and continues waiting for a response from the slave. In the **WAIT_FOR_OK** mode, the master has told the slave that it should sleep and is waiting for a **SLEEP_OK** message.

When the master is in one of the final set of states (**SENDING**, **SEND/RCV**, and **RECEIVING**), it is actively sending and/or receiving data. In the **SENDING** mode, the master may receive a **NEW_DATA** message from the slave. The master responds with a **WAKE_UP** message and enters the **SENDING_WAIT** mode. When neither the master nor the slave have any more data to send, the master sends a **SLEEP** message and enters the **WAIT_FOR_OK** mode.

Wireless connections are very susceptible to interference from both external devices and other wireless devices using the same settings or talking to the same base station. By using this protocol, we provide the base station with useful information about the communication patterns of the mobile host. Although not required by the protocol, the master can inform the slave of its sleep time, or the slave can suggest appropriate sleep times to the master. If the protocol is used such that only prespecified timeouts trigger restarting communication, the slave can design a communication scheduling algorithm based around the known sleep time of the master. Additionally, if the sleep times for the master are sufficiently long, the slave can save any data destined for the master to disk. This will free the buffer space being used by the data destined for the master so it can be used for other active communications.

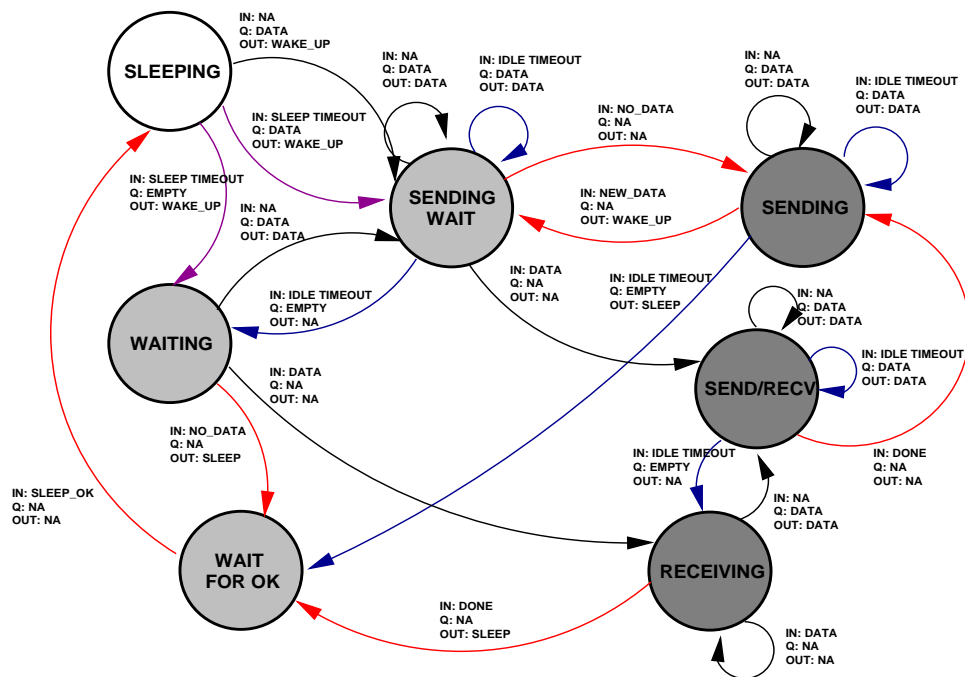


Figure 11: Master (Mobile Host) Protocol State Diagram

4.3.2 Timing Considerations

Timing is a key issue for both the performance of the mobile host as well as the amount of energy that can be saved. If the wireless Ethernet card is suspended too often, the user will see lags in data transfer performance. On the other hand, if it is not suspended long enough, the gain in battery life time may be undetectable.

In order to determine when the card should be suspended, the protocol needs to determine the communication patterns for both sender and receiver. There are two ways by which idle periods in the communication can be detected. The first, and simplest, is when the application can actually inform the protocol that it doesn't have any data to send. This requires a more complex application that has information about its communication patterns. The second method is to use a timer set with a *timeout period*. If the timer expires and no communication has occurred since the last expiration, the protocol concludes that there is an idle period in the communication. The appropriate timeout period depends on the requirements of the application. Timeout periods that are too short may cause the protocol to go to sleep prematurely, resulting in poor response time for applications dependent on communication. On the other hand, timeout periods that are too long may cause the protocol and the communication device to remain active for unnecessarily long periods of time, wasting precious energy. The communication layer can reconcile these tradeoffs by using the payoff curves for delay and energy consumption provided by the application.

The other timing parameter is the *sleep duration* which defines how long the master should keep the communication suspended. The appropriate sleep duration also depends on the requirements of the application. Longer sleep periods will cause longer lags in any interactive applications. Shorter sleep periods will not extend battery lifetime appreciatively. The application needs to determine

the appropriate tradeoff for battery lifetime versus delay. In many instances, the expected time and data size for the response to a request initiated by the mobile host can be estimated. This includes, for example, applications like mail, web browsing, and file transfer. In this context, hints provided by the application could be very helpful. In our experiments reported in Section 4.4, we examine the effects of fixed timeouts that require no application support and can be implemented within the transport layer. Adaptively varying the timeouts or using learning techniques are discussed in Section 4.5.

A mobile host that is running multiple applications cannot base its power strategy on the expected communication patterns of a single application. In this situation, the power management protocol must take hints about sleep/wake up durations for all executing applications. By exposing power management to the application, and hence to the user, our power management protocol can be guided in the appropriate allocation of resources.

A final consideration is the time required to wake up and shut down the specific wireless network card. Our protocol is designed to be independent of the specific card being used. Since our techniques address issues regarding the end-to-end transmission of data, we assume that this wakeup time is minimal in comparison to the total transfer time. Although this may not be true for all devices currently, the interface standards proposed in [33] suggest that future devices will provide relatively inexpensive transitions between waking and sleeping states.

4.4 Experimental Evaluation

The goal of our experiments is to show that, by using our power management techniques, we can save a significant amount of the energy consumed by the wireless Ethernet card. The tradeoff is an increased transmission delay observed by the receiver. First we will present our experimental setup and the user communication patterns used in our experiments. We will then show the impact of the power management techniques in the context of these user communication patterns. Finally, we will discuss our results in the context of several real systems.

4.4.1 Energy Measurement

In order to understand how to accomplish power management for communication actions, we first need to define what we mean by the amount of energy consumed by each such action. Previous work in this area has defined this to be the amount of energy consumed by the network interface device itself [17, 64, 72]. We extend this definition to consider the energy consumed by the entire mobile host, which includes both the network interface and the rest of the system involved in the communication. Thus, our measurement model evaluates the total energy consumed by the entire system, rather than focusing on the requirements of individual devices. In effect, we are “charging” (no pun intended) the communication action for all system usage that occurs during that action. This is conservative, but is realistic for many applications as long as communication delays remain short enough so that it is not practical to browse the Web, for example, in parallel with some other activity.

Figure 12(a) shows a sample energy measurement for a mobile host while the machine is idle, with and without an attached network interface device, and while the machine is actively communicating. Three sources of energy consumption may be differentiated in Figure 12(a): communication-specific, device-specific and machine-specific. The area under the solid line in the first time period represents the energy consumption of the idle machine, including the CPU, disks, and other components besides the network interface device. This is the *machine-specific*

base amount, indicated by power level A. This base amount will always be consumed. During idle times, this amount may be reduced with machine-specific power management techniques, but we assume that while communication is ongoing such techniques will not be invoked. In the next time period in the figure, a wireless network interface device has been activated. Therefore, this area represents the energy consumed by both the idle machine and the idle wireless network interface device. This *device-specific* energy consumption by the network interface device is indicated by the section between power levels A and B. As with the machine-specific power management techniques, device-specific techniques may be used to reduce energy consumption during idle periods in communication. Since we are targeting active communication, we can assume that the device must be turned on during those periods and will consume some average amount of device-specific energy. Finally, when the mobile host is actively communicating, as indicated by the values above power level B, the energy consumed is *communication-specific*. This energy includes the energy consumed by the wireless network interface device for data transmission as well as any energy consumed by the CPU (and other devices involved) during data and protocol processing for transmission and reception. Since the use of the hard disk during data transfer is highly dependent on the application being used, for the remainder of this chapter, we assume that the hard disk is not involved in communication. This assumption allows us to focus on the CPU and network interface device. It would be interesting in the future to investigate the interaction between power management techniques for active communication and power management techniques for hard disks.

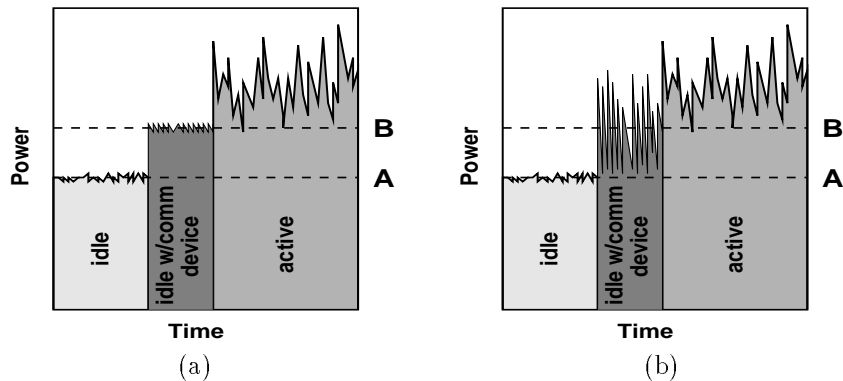


Figure 12: Energy Components

Our measurement model is derived from the actual measurement of energy consumed by the whole machine. By doing this, we limit ourselves from being able to single out the energy consumption of individual devices, but instead, we are able to consider the energy consumption of actions performed on the machine. Figure 12 shows two sample power measurements for a mobile host. Figure 12 (a) represents no power management and Figure 12 (b) represents the effects of power management.

In this chapter, we aim our efforts at reducing the amount of energy consumed when the communication is idle. From the power management techniques reflected in Figure 12(b), we see that there is a fluctuation of the amount of energy consumed during idle periods in the communication. The lower levels of energy consumption represent the effort of our power management techniques. The spikes in the energy consumption represent the overhead, both CPU energy consumption and communication device energy consumption, during our power management efforts. For all of the experiments described in the next section, we consider the energy consumption above

power level A to be attributed to communication.

This view of power management allows the individual devices to perform their own power management techniques. For example, our techniques do not determine when to suspend the CPU or how the wireless communication device should manage power during active communication. Instead, we are concerned with reducing the energy attributed to ongoing communication. Therefore, we do not try to separate the energy consumed by the CPU during protocol and data processing and power management from the energy consumed by the wireless communication device.

4.4.2 Experimental Setup

In order to determine the impact of our power management techniques, we measure the energy consumption of a wireless Ethernet card under varying conditions. In our experiments, we use a 915MHz Lucent WaveLAN PCMCIA wireless Ethernet card that can transmit data up to 150KBps. It provides three power modes: transmit, receive and suspend, and does not perform power management at the MAC layer. The system is configured as shown in Figure 13, with a wireless Ethernet in a NEC Versa 6320 laptop (the mobile host) communicating with a Gateway Solo 2200 (the base station) using a second WaveLAN PCMCIA card, both machines running Linux. We plugged the laptop into a universal power supply (UPS) to filter out any fluctuations in the wall voltage. Our multimeter samples the current 11-12 times a second. From these samples and the output voltage of the UPS, we can monitor the power being used by the computer.

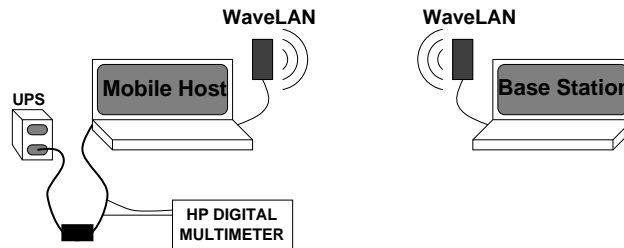


Figure 13: Experimental Setup

To determine the mobile host’s energy consumption, we monitor the current being drawn from the transformer by the host. Figure 14 shows the output of the multimeter over time for a no power management example and for a power management example running on the NEC Versa. This trace of current readings (11-12 readings a second), when integrated over time, provides us with the total energy and average energy consumed during that time period. In Figure 14, the solid line near 14W represents the power measured for this specific computer when it is idle. In this situation, we define idle to be not communicating, but also not invoking machine-specific power management techniques. From this baseline information about the necessary energy to run a computer, we can compute the “cost” of communication. This cost includes the energy consumed by the communication device and any energy consumed by the CPU due to the communication. Each experiment was performed over a period of 30 minutes to provide sufficiently long samples. To ensure stability in our reported numbers, we repeated our experiments several times for each scenario. The results presented in this section were taken from specific sample runs. Each individual run was chosen from a set of qualitatively similar runs of a particular experiment.

According to specifications from the manufacturer [46], the energy requirements of the WaveLAN card are those shown in Table 1, Column 2. Column 3 in Table 1 shows the energy requirements measured during our experiments. The measurements for receive mode are taken while the

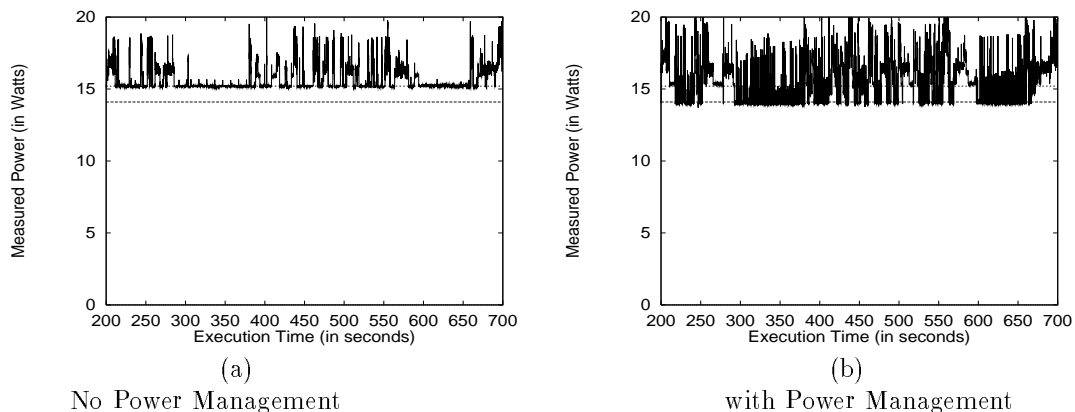


Figure 14: Sample Output from Multimeter

computer was idle, which implies no extra disk or CPU activity. We observe that our measurements of the energy required while the device is in either mode are very close to the documented specification.

State	Documented	Measured
WaveLAN - suspended	0W	0W
WaveLAN - receive	1.48W	1.52W
WaveLAN - transmit	3.00W	3.10W

Table 1: Power Requirements of the Lucent WaveLAN PCMCIA Wireless Ethernet card.

For an example of the different levels of energy consumption, again consider Figure 14, which shows sample traces for the the energy consumed over a period of time for the NEC Versa. From our measurements, we determined that the power for the laptop measures around 14W when idle. With the WaveLAN card inserted, the power for the laptop measures 15.5W when idle. From this information, we consider any power over the 14W idle to be contributed to the ongoing communication. Our goal is to conserve as much of the 1.5W for the idle communication device. Additionally, we can see that the power often peaks over 17W, which is the expected power for a transmission. This additional power over 17W can be attributed to the use of the CPU during active data transmission. As mentioned earlier, we also consider this power to be part of the overall communication energy consumption.

We chose Linux as a research platform because of the available source code for both the PCMCIA driver and the WaveLAN driver. In order to suspend the WaveLAN card in Linux, a system call to the kernel is used to send a suspend command to the WaveLAN driver. Suspension stops the receive unit, turns off the card, and updates the status of the PCMCIA device, removing its entry from any routing tables. This update generates an unwanted disk access. We modified the WaveLAN driver to update the status, but to leave the routing tables untouched, and called this the “sleep mode”. Switching from active mode to sleep mode is now a matter of only the system call to the kernel and does not access the disk. Similarly, to wake up the WaveLAN card, a system call to the kernel is used to restart the receive unit on the WaveLAN card. The next generation

of WaveLAN cards will have a DOZE mode [35] that will provide a quicker transition from active to DOZE than the transition from active to suspended in the current model. Our power management protocol was implemented in the context of an adaptive communication framework that provides dynamic protocol configuration support to the application [40, 38]. Through the use of the framework interface, the application can set and change specific protocol parameters.

4.4.3 Communication Patterns

The effectiveness of any power management strategy is dependent on the usage patterns of the target of such strategies. In the context of communication-based power management, we want to explore users with varied communication demands. For our experiments, we distinguish between two different types of communication patterns. The first, simulation-based, simulates a user from a general profile of their communication usage. The second, trace-based, emulates the communication usage of an actual web user.

4.4.3.1 Simulation-based Communication Patterns

The communication patterns used in our initial experiments were designed to simulate different types of users. The amounts of data transmitted and received and the idle patterns of the users are varied randomly over time. The communication patterns are chosen to model three “typical” users. Table 2 presents the minimum, maximum and average amount of data in a transmission for each of these users. For the simulated web users, a transmission from the mobile host triggers multiple responses from the base station to simulate the multiple files needed for a web page. The average number of responses is shown in the count column of Table 2. Table 3 shows the timing patterns for the same users. Each user is described by a transmission cycle. During this cycle, the mobile host transmits and receives the amounts of data as described in Table 2. When considered with the idle times shown in Table 3, we can see the total amount of time per cycle that the mobile host spends transmitting, receiving and sitting idle. Our goal is to suspend the communication device during as much of these idle times as possible.

The first pattern (WEB) simulates a user browsing the web. The amount of data transmitted is relatively small in comparison to the amount of data received. The sleep time represents the amount of time the user would spend reading a page before going on to the next one. Each request transmitted by the mobile host triggers a number of responses. The delay between these responses is varied from 0-15 seconds to simulate a busy web server. The second pattern (JW) simulates a user working on a joint project. This user occasionally transmits and receives large amounts of data. There is no connection between transmissions from the mobile and from the base station. The third pattern (EMAIL) simulates a user that mostly transmits and receives e-mail messages. This user is idle most of the time between transmissions, and transmissions size is relatively small.

Typical mobile users tend to perform each of the above activities to some degree. From that point of view, the patterns presented above categorize users according to their main activity. The goal of the adaptive techniques discussed in Section 4.5 are to dynamically find appropriate timeouts for user performing such activities.

4.4.3.2 Trace-Based Communication Patterns

Our second set of communication patterns was derived from traces of active web users obtained from a Lucent/Bell Labs web proxy server. We use these simulations to demonstrate the effects of power management on users performing real actions. Additionally, we were able to use traces

Pattern	Data Transmitted					Data Received				
	Min (KB)	Max (KB)	Avg (KB)	Count	Avg Total (KB)	Min (KB)	Max (KB)	Avg (KB)	Count	Avg Total (KB)
WEB	5	30	17.5	1	17.5	300	1200	750	10	7500
JW	5	500	227.5	1	227.5	5	500	227.5	1	227.5
EMAIL	5	300	152.5	1	152.5	5	300	152.5	1	152.5

Table 2: Data Communication Patterns for Three Simulated Users

Pattern	User Sleep Time			Average Time			Average Percent Sleeping
	Min (sec)	Max (sec)	Avg (sec)	Transmitting (sec)	Receiving (sec)	Sleeping (sec)	
WEB	10	300	155	0.116	50	104.9	67.7%
JW	10	300	155	1.52	1.52	151.96	98%
EMAIL	10	600	305	1.02	1.02	302.96	99%

Table 3: Timing Patterns for Three Simulated Users

depicting different levels of web activity. For example, Figure 15 shows two traces of the amount of data received over time for our web user. The first is a low-end user trace where about 2MBytes of data is received in an hour. The second is a high-end user trace where about 125MBytes of data is received in an hour. For clarity, the scale of the high-end user trace graph in Figure 15 (b) was reduced to allow for comparison between the two traces. The high-end user trace includes a number of received transmissions ranging up to 16MBytes not shown in the graph. Additionally, we consider a middle user trace where about 64MBytes of data is received in an hour.

4.4.4 Results

In this section, we consider the results from our experiments in the context of the effects of our power management protocol on the energy consumed by the communication. This energy consumption is affected by the use of the CPU and the hard disk during communication. The savings we see come predominantly from the reduced consumption by the wireless Ethernet card. In Section 4.4.5, we discuss our results in the context of three real systems. The effects of the power management on a real system will depend on the power requirements of the system itself.

4.4.4.1 Protocol Energy Consumption

When running the experiments with our management techniques turned on, we incur some overhead in terms of energy consumption. During idle periods in the communication, the overhead is due to the cost of waking up the WaveLAN card, transmitting a query to the base station and putting the card to sleep immediately since there is no data to receive. Our results show that, even with relatively short sleep times, this overhead is still significantly less than the energy consumed by the WaveLAN card had it been left in receive mode.

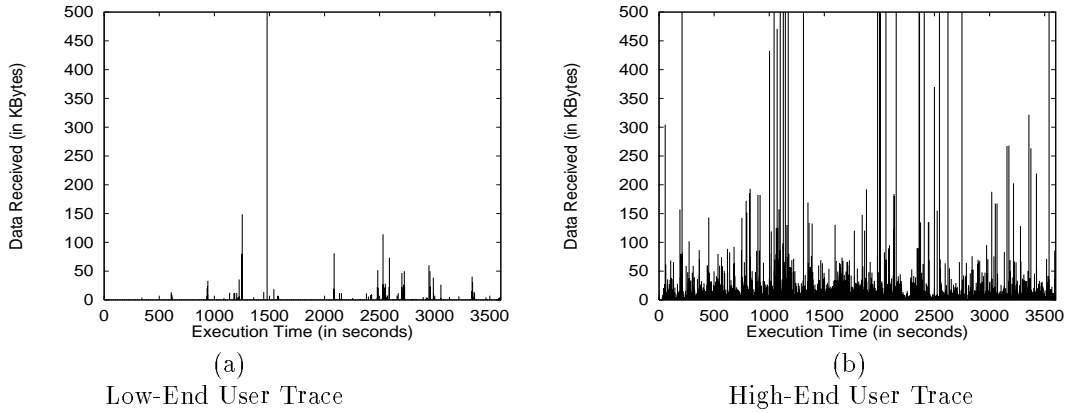


Figure 15: Trace-Based Data Usage

Our experiments produce a trace of the power measurements from the multimeter. Plotting these traces gives us a good, intuitive understanding of the effect of allowing the wireless Ethernet card to sleep for short periods of time. In each graph in Figure 16, we compare two traces of the power for the idle communication subsystem (i.e., when there is no actual transmission or reception). In Figure 16a, the sleep duration is 1 second, and in Figure 16b, the sleep duration is 2 seconds. The first trace, the flat line at 1.5W marked by the diamonds, shows the power for the communication when no power management is performed. The second trace, the line at 0W with occasional spikes marked by the plus signs, shows the power readings with our power management protocol turned on.

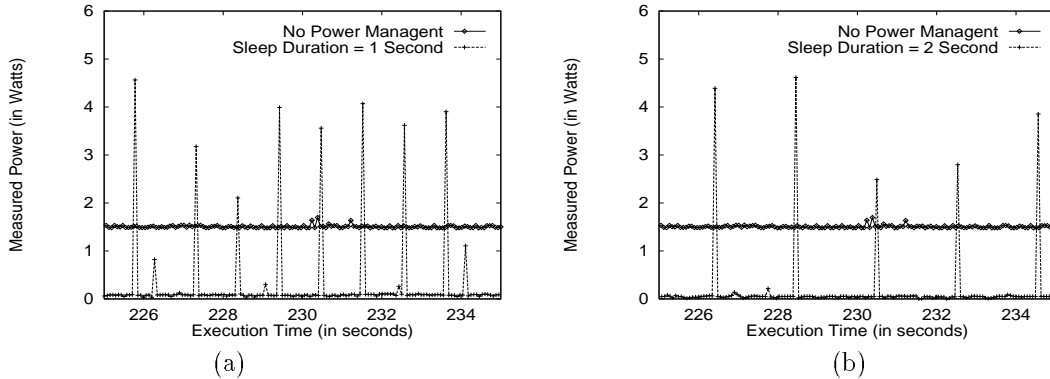


Figure 16: Power Readings During a Sample Idle Period

We can see from the two traces that the power is approximately 1.5W less when the WaveLAN card is suspended. For the first trace, the power stays at 1.5W since the communication device is always powered on. For the second trace, the power is near zero most of the time, but spikes up at regular intervals. These spikes are caused by the protocol waking up and sending a query to the base station to see if there is any data waiting to be sent. By comparing the two graphs, we can see that the overhead for transmitting the queries is going to increase as the sleep duration gets shorter. Figure 17 shows the percent savings of using our protocol during idle periods when

compared to no power management.

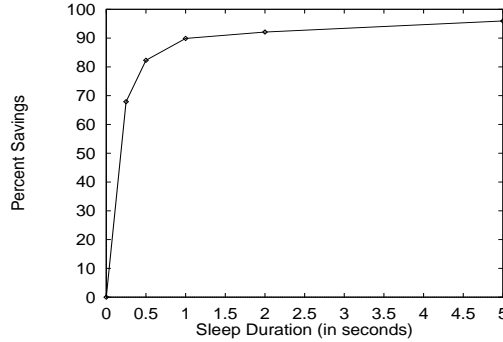


Figure 17: Energy Savings During Idle Periods

Figure 18 shows a trace during the transmission of a message to the base station. As we can see, the energy consumption for both traces is the same during the transmission. The difference lies in the fact that after the transmission is complete, the trace using power management shows the effect of suspending the wireless Ethernet on the energy consumption.

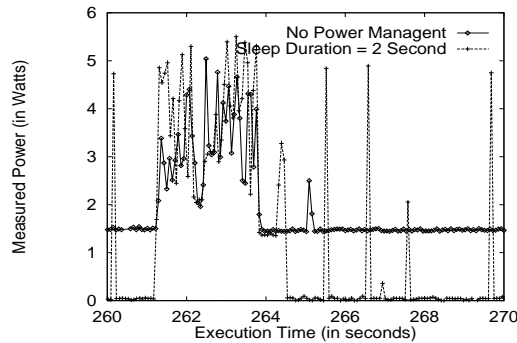


Figure 18: A Sample Transmission

4.4.4.2 Energy Savings for Communication Patterns

In order to determine the longer term effects of power management, we measure energy consumption during communication generated in the patterns discussed in Section 4.4.3. In our experiments, we do not queue data at the master; i.e., we always wake up the communication device at the master if there is data to send.

Simulation-based. Figure 19 shows the results from our experiments in terms of the percent of the total energy consumed by the communication that was saved by using our power management protocol. For WEB, we see a 48-57% savings in the energy consumed by the communication. For JW, we see a 54-78% savings in the energy consumed by the communication. In the case of EMAIL, we compare the results of no power management with those of power management with sleep durations of 1 and 5 minutes. (Due to the different time scale, these measurements are not included in Figure 19.) These sleep durations result in a savings of 81% and 83% of the energy consumed by the communication.

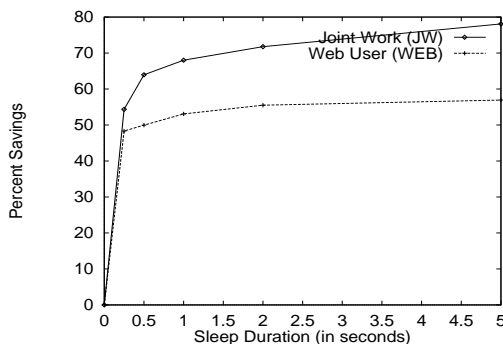


Figure 19: Percent Savings for Communication Energy Consumption for Simulation-Based Communication Patterns

Trace-Based. Similar to the results for the simulation-based communication patterns, Figure 20 shows the communication energy savings for the trace-based web users and also includes the energy savings for our simulation based web user. As would be expected, the amount of savings increases as the user becomes less active. It is interesting to note that the amount of savings for the different users levels out in different places. For the high-end user, we don't see much of a savings improvement after a 0.5sec sleep duration, while for the low-end user, the savings increases significantly up to a 1sec sleep duration. This should be expected, since the high-end user continually sends significantly more data, allowing for less opportunities to sleep for long periods of time. It is interesting to observe that the power management results for our simulated WEB trace compares closely with a middle web user trace.

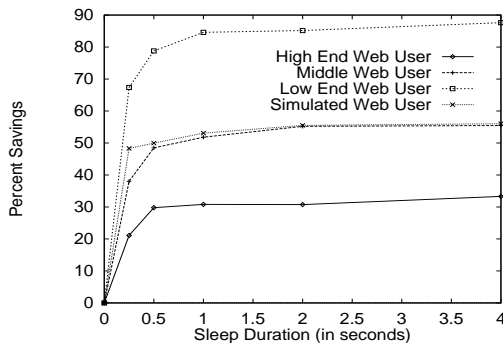


Figure 20: Percent Savings for Communication Energy Consumption for Trace-Based Communication Patterns

4.4.4.3 Delay

Energy savings never come for free. In the context of communication, this cost can be measured in delay. A sleep duration of any length will impose, on average, a delay of half the duration. This cost must be taken into consideration when deciding how much power management to use. In the context of a user that solely uses communication for email, a sleep duration of 1 minute

is acceptable. In reality, many email programs check mail on the order of every 5 minutes, and hence, such a sleep duration is more representative of the common email user. In contrast, a user who is working jointly across the network or who is accessing web pages may not be willing to accept such delays. In these cases, the sleep durations should reflect the tolerance of the users to delays in receiving data.

In the context of these experiments, we measure the communication delay in terms of additional transmission time per user data block at the base station. (We never queued data at the mobile machine.) We calculate this delay by determining the amount of time that the first message in the data block was delayed. Once the first message is sent, the rest will follow, and the delay to those messages will depend purely on how quickly the data can be transmitted. It is easy to show that the maximum additional delay imposed by our protocol on any data can never exceed the delay for the first packet, and hence, the numbers we present are a conservative estimate. We measure this delay by determining the time the transmission request was sent to the communication subsystem and the time when the data is finally sent. Since multiple transmission requests may queue up at the base station during large bursts of traffic, delays may be incurred even when no power management is used. Figure 21 shows the average added delay for a given sleep duration for the trace-based web patterns. As the user becomes more active, the average delay decreases due to the fact that the communication is suspended less often, allowing data transmissions to be sent back-to-back without extra delay.

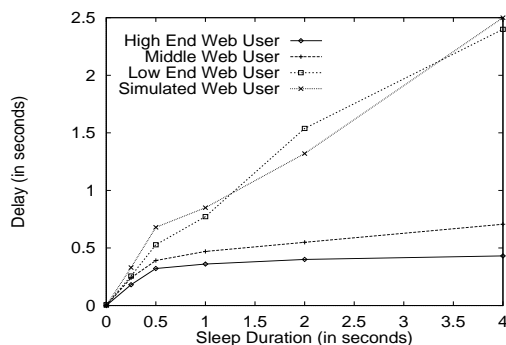


Figure 21: Average Added Delay based on Sleep Duration

Figure 21 also includes the average added delay for the simulation-based web user. Although we have already discussed the fact that the energy savings for this communication pattern matches that of the middle web user, the delay observed for this pattern is closer to that of the low-end web user. This inconsistency is due to the differences in the implementation of the two types of experiments. In the case of the trace-based web users, data transfer was performed in a request-response fashion, where each request resulted in one response. Since the user was communicating via a web proxy, requests often resulted in immediate responses, and so were not delayed by the power management protocol. In the case of the simulation-based web user, a single request resulted in potentially multiple responses. These responses were timed to simulate communication delays to a potentially busy server, and so were rarely able to send data quickly enough to return data before the power management protocol went into a sleeping state.

4.4.5 Impact on System Costs

Now we consider the energy savings in the context of three real machines (see Table 4). The experiments that we have described were performed on the NEC Versa 6320. We also measured the idle energy consumption for the Toshiba Libretto 60 with and without the WaveLAN card and the idle energy consumption for the HP Palmtop PC 320LX. We then used the results from Section 4.4.4, running the experiments on the NEC, to estimate the effect of the communication patterns on the other two machines. The Toshiba Libretto 60 is a small laptop that can run either Windows 95 or Linux. We would expect similar power costs for the hard disk and CPU for this machine as we did for the NEC. The HP Palmtop PC 320LX, on the other hand, runs Windows CE and has no internal hard disk. This will probably change the effect of power management on this type of machine. It may be argued that a wireless card with the power profile of WaveLAN is unlikely to be used with an HP Palmtop-like machine. We simply use this as an example of the trend towards lighter, more power efficient machines that have small battery capacity.

Machine	Power Requirements	
	Idle w/o WaveLAN Card	Idle w/ WaveLAN Card
NEC Versa 6320	14W	15.5W
Toshiba Libretto 60	7W	8.5W
HP Palmtop PC 320LX	1.2W	2.7W

Table 4: Measured Power Requirements for Three Machines

The first machine, the NEC Versa 6320, is a high end machine that consumes 14W while sitting idle. In this context, the 1.5W consumed by the WaveLAN card only represents approximately 10% of the power drawn by the computer when it is idle. In comparison, a machine like the Toshiba Libretto consumes only 7W when idle. The 1.5W of the WaveLAN card now represents approximately 18% of the power drawn by the computer when it is idle. If we take this one step further, we can see that for a machine like the HP Palmtop PC, this percentage increases to over 50%. To compensate for this problem, some manufacturers have introduced wireless Ethernet cards that have an internal battery, although they still draw some amount of power from the main battery. In this case, the power drawn from the main battery by the idle device represents approximately 10% of the entire system power of the HP. Although this reduces the effect on the lifetime of the main battery, we still need to consider the effects on the battery for the card itself. Our techniques will work to extend the lifetimes of both batteries.

Considered in the context of the NEC, the results discussed in Section 4.4.4 show a 6.2-8.9% savings for the JW pattern and a 8.0-9.5% savings for the WEB pattern. If we now project these results onto the other two machines, we see even better savings of the energy consumed by the entire system. Figure 22 compares the results for the percent saved of the total system energy for these three machines over varied sleep durations. The crossing of the plots for the WEB and JW patterns for the HP is due to the fact that the energy consumption of the communication device is more than the energy consumption of the HP. Therefore, the better communication energy savings for the JW pattern dominates the results. This is simply an artifact of the fact that the two patterns will eventually cross for all cases, as can be seen by the plots for the Toshiba. Most importantly, we can see that the trend toward machines like the Toshiba and the HP make it

imperative that we properly manage communication devices, since the energy consumed by these devices represents more and more of the total system energy.

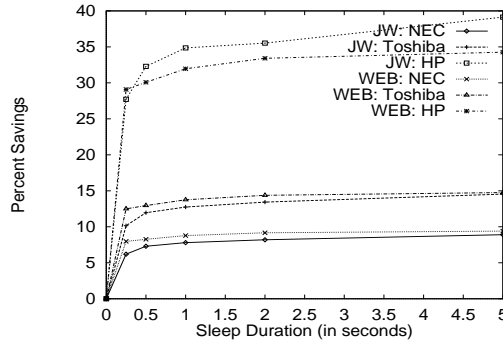


Figure 22: Percent Savings for Three Types of Machines

4.5 Adaptive Mobile Power Management

We have shown that power management for communication devices can extend battery life. We also see that there is not one scheme that will fit all users or all applications. This leads us to investigate mechanisms for adapting power management levels during communication. The goal of this chapter is not to address the issue of prediction, but provide the mechanism by which predictive algorithms can be used to adjust power management parameters; in particular, the timeout and sleep duration in our implementation.

The ideal power management technique would sleep whenever there is no data to receive from the base station and wake up for any incoming receptions as well as tell the base station exactly when to expect transmissions from the mobile host. The goal of adaptive power management techniques is to be able to estimate when there is data to transmit from either side. Poor prediction can cause unsuccessful power management and waste resources such as buffer space and bandwidth.

With our protocol, the sleep duration can be adapted to fit the communication patterns of the application. As the sleep durations increase, we can see that the curve for the amount of energy saved will level off. This happens as the savings reach the theoretical maximum savings for the particular communication pattern. The theoretical limit is reached when the communication device is only active when there is actual data transfers occurring. For smaller sleep durations that are much smaller than the expected time between transmissions, the communication device is still active during some of the idle period. As the sleep duration increases, the probability that there is data waiting at the base station increases. As an example, consider Figure 23 which shows the percent of the total time spent sleeping for both the JW and the WEB patterns. As the sleep duration increases, the percent of the total time spent sleeping approaches the theoretical limit (98% for JW and 67% for WEB; see Table 3). From the fact that the sleep time for the WEB pattern comes closer than the JW pattern to the optimal sleep time, we can see that our power management techniques were more successful for the WEB pattern.

By providing an application-level interface to our power management protocol, the policies used for determining sleep durations can be controlled. In this way, application-specific information in the form of payoff functions can be used to determine optimal adaptation strategies.

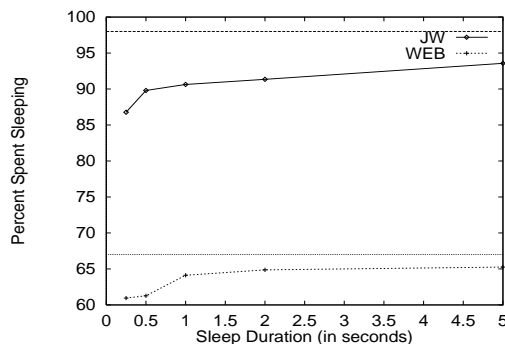


Figure 23: Percent of Total Time spent Sleeping

In the context of our experiments, we implemented a simple adaptive algorithm similar to [24] for the web user. The algorithm responds to communication activity by reducing the sleep duration to 250 milliseconds and reacts to idle periods by doubling the sleep duration up to 5 minutes. We can use this simple algorithm because the communication patterns of the web user are somewhat predictable. A request from the mobile host expects multiple responses from the base station. As the mobile host notices that no more responses are available, it can deduce that either there are no more responses or that the server is busy and the responses may be delayed. For this communication pattern, such an adaptive algorithm provides a 58% savings in the energy consumed by the communication device. This savings is an improvement over the 5 sec static sleep duration. While for the 5 second sleep duration, we saw a 3.12 second additional delay, for the adaptive algorithm, we only saw a 2.77 second delay. This suggests that adaptive and predictive techniques have merit in mobile communication power management for communication applications. It also demonstrates that applying power management at the transport or application layer has benefits. These techniques can be used in conjunction with adaptive techniques used at the MAC layer [17, 49].

An important aspect to keep in mind is that efficient prediction or estimation is not always simple or useful. Taking the communication patterns of multiple applications would also make adaptation more challenging. Learning-theory based estimation techniques [42, 36] can provide better adaptive algorithms for deciding when to power off and when to turn back on the communication device. Many such techniques inherently try to estimate the distribution generating the communication packets, and hence application provided hints help such estimation techniques.

4.6 Overview of Contributions

In this chapter, we have explored the relationship between two resources that are important to the application: service quality and power. Exploring and understanding the relationship between these very different resources gives us the ability to better serve the application. Traditional approaches to resource management cannot handle such diverse resource definitions. By using the abstraction provided by payoff functions, the cooperative communication layer can optimize performance across power and service quality.

Specifically, we have presented a novel transport-level protocol by which a mobile host can judiciously suspend and restart its communication device, and by informing the base station appropriately, not lose en-route data. We have presented experimental results from an implementation of this protocol, and shown energy savings of up to 83% for communication. This translates to a savings of 6-9% in terms of total system energy for high end laptops, and can represent up to 40% savings for current hand-held PCs. When we consider the subset of our results for email and web browsing applications in the context of hand-held PDAs, our implementation results agree in large measure with the simulation results in [65]. It is important to note that if other components of the mobile machine are managed better, the relative improvement numbers due to efficient power management of the communication device will be more prominent. For most applications the resulting small additional delay (e.g., 0.4-3.1 seconds for some web browsing responses) should be acceptable.

Chapter 5

Variable Reliability Protocol

Historically, applications had to choose between completely reliable message transfer (as with TCP) and “best effort”, non-guaranteed service (as in UDP). This presents little choice to applications able to handle relaxed levels of reliability. Instead, applications may have to pay the price of using a reliable protocol, which can be measured as the unnecessary retransmission of messages by the sender and as the amount of buffer space for buffering messages at the sender and the receiver. Alternatively, applications may use unreliable protocols, which may result in unacceptable losses.

This chapter presents a protocol for implementing variable reliability communication. The premise of this work is that many multimedia applications will benefit from being able to choose between different reliability levels, rather than being forced to choose between full or no reliability. By using our variable reliability protocol, applications can explore tradeoffs between data reliability and transfer time. Examples of the use of this protocol is quality-based partitioning of data and adaptation of reliability based on the current state of the application and the network. We present results from our implementation and demonstrate the usefulness of adaptive reliability variation for sample distributed applications.

This chapter describes the definition of a variable reliability protocol for a specific reliability stream. In order to enable applications to switch between different reliability levels, the variable reliability protocol can maintain multiple reliability streams in parallel. Each of these reliability streams corresponds to a virtual communication channel, as described in Section 2.1. The sender can specify any one of these streams for each data message it sends. In this way, the sender determines what reliability is required for specific messages, and the receiver implements the policies for each reliability stream. For a specific example, consider the different types of frames in an MPEG video stream. Each type of frame can be transmitted via a different reliability stream, thus allowing I frames to use the highest reliability stream while still allowing P and B frames to use middle and lower reliability streams. A specific example of the use of this protocol is presented in Chapter 6.

Our techniques are evaluated in the context of a distributed virtual environment. The protocol discussed in this chapter address the reliability requirements of large data sets within such an application. Specifically, some of the graphical objects in the environment are “continuous objects” in the sense that their display requires continuous updates to images or texture maps or even motion JPEG. This chapter first defines the problem within the context of the reliability of data transfers. We then specify and explain the details of our variable reliability protocol.

5.1 Reliability and Communication

It is well-known that for multimedia applications the two types of reliability provided by TCP and UDP are insufficient. This is due to the fact that most such applications are able to tolerate some

application-specific losses. Given a lack of choice, such applications must use reliability even when they do not need it or implement their own versions of reliability on top of an existing unreliable protocol. [47] presents a way to quantify the costs of “too much” or “too little” reliability within the specification of their protocol. Similarly, [20] evaluates the cost of using standard reliable sliding window protocols like go-back- n and selective retransmission in situations where reliability could be relaxed. In comparison, our approach is to provide a framework in which each application can define its own notion of reliability. Additionally, we allow the application to define different levels of reliability for data with different requirements.

Implicit in our design is the concept of *application layer framing* (ALF) [18]. Namely, we assume that an application can deal independently with the data units it is sending. In other words, each unit of data sent by the application contains sufficient identifying information to allow the application to process it. Among other things, this means that data need not be held up for ordering constraints (beyond those defined below).

Reliable data transfer can be defined as a service that guarantees the delivery of data sent from a sender to a receiver without duplication, loss, or out of order messages. Unreliable data transfer makes no guarantees as to duplication, loss, or order. It is not obvious how to define a service “between” the two extremes of reliable and unreliable data transfer. The following sections discuss the issues involved in defining such a service and describe a specific implementation of a variable reliability protocol.

5.2 Specification of Variable Reliability without Timing Constraints

Variable reliability mechanisms can be applied to realtime and non-realtime data. This section addresses non-realtime data (i.e., data that is time-sensitive, but not time-critical). [20] and [53] address reliability for realtime continuous traffic streams. This difference in focus permits us to ignore lifetime issues for individual messages and concentrate on issues concerning the amount of data received and the spacing of message losses. We target applications where the structure of the data is important and where timing issues are based on frame rather than individual message transmission times. This type of specification can be particularly useful for applications running over low-bandwidth and/or high error rate networks (e.g., wireless networks or the Internet). In the next section, we discuss an extension to our variable reliability protocol to including timing issues.

The scheme we specify below defines a service based on simple loss measurement and retransmission policies. The protocol specified in [47] provides probabilistic reliability guarantees based on the number of retransmissions of a message. In contrast, our protocol provides applications with hard guarantees about reliability based on specified loss allowances. The following describes some parameters of our variable reliability protocol.

5.2.1 Loss Tolerance

Two mechanisms govern loss tolerance for our variable reliability protocol. The first is the definition of *application-specified data boundaries* or *data frames*. A data frame consists of data messages, each of which is an *application data unit* (ADU). The messages from different frames are handled separately, thereby limiting the effects of problems in the transmission of one frame on the transmission of others. Since a data frame can also be considered an ADU, this partitioning of data frames into data messages provides a two level hierarchy of ADUs, where the ADUs in

each level can still be process independently of other ADUs of that level. The second mechanism is a simple *sliding window* within the messages of the data frame. The use of these mechanisms is controlled by the specification of suitable loss tolerance parameters.

Two parameters may be specified by the application at the data frame level. The first parameter is the *maximum number of consecutive losses*, which essentially defines the tolerable ADU loss burst size. The second parameter is the *high level loss percentage*, which indicates how much loss in the total data frame the application can tolerate. At the sliding window level, the application can define the *maximum number of losses allowed in a window*. These parameters are similar to those specified in [27].

In Figure 24, the maximum number of consecutive losses is 2, which means that there will never be more than two consecutive losses; the high level loss percentage is 50%, which means that the receiver is guaranteed to receive at least 50% of the ADUs in each frame; and the maximum number of allowable losses in a window of size 6 is 3. The combination of maximum number of allowable losses and maximum burst size ensures that the losses are spread out through the messages of the data frame and not grouped together in one portion of the frame.

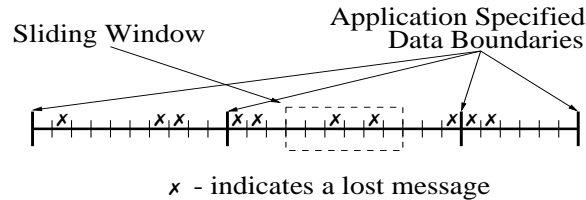


Figure 24: Example of Allowable Loss in a Sliding Window and in a Data Frame

5.2.2 Message Order

Applications can specify the degree of ordering required within a data frame. Since we are talking about variable degrees of reliability, this may include gaps in the sequences of messages. The *ordering distance* parameter defines the ordering specifications for the protocol by specifying when new messages can be passed up to the application. If the *ordering distance* is 0, No messages will be delivered out of order. This implies that if a message is received that violates this guarantee, it will be dropped and not passed up to the application. For example, if it is acceptable to lose three out of ten messages, the messages delivered may be the following:

0, 1, 3, 4, 5, 8, 9

Although not all of the messages were delivered, no message was delivered before an older message. If in this example, message 2 arrived after message 3 had been delivered to the application, message 2 would be discarded. If the *ordering distance* is infinity, the protocol does not enforce any type of ordering constraints on the data. Data is passed up to the application as soon as it is received. If the *ordering distance* is n , the receiver will never accept a message that is more than n distance from the newest message. For example, if it is acceptable to lose three out of ten messages, and maximum misordering distance is 2, the messages delivered may be the following:

1, 0, 5, 3, 4, 8, 9

The delivery of message 5 to the application guarantees that message 2 will never be delivered (since it is more than 2 older than 5).

5.2.3 Application-Specified Data Boundaries

The application can specify how many data frames it can handle simultaneously. The protocol will then guarantee that the data being passed to the application will always belong to appropriate frames. Given that some number, n , of frames may be transmitted simultaneously, we define a sliding window on data frames. If the sliding window size is n , at most n frames will ever have ADUs in transmission at any given point in time. The sender will not start to send the $n + 1$ st frame until the first frame has been “successfully” transmitted, where “successfully” is defined in accordance with the specified reliability. Messages from old frames will be discarded on the receiving side.

5.2.4 Protocol Specification

The following provides a description of the API for the variable reliability protocol without timing constraints.

Parameters : First we will describe the set of parameters that are available to the application. These parameters represent four areas: Frames, Ordering, Loss Tolerance and Control.

Data Frame Parameters : FRAMEPARAMETERS

FW : *frame boundary window*. This parameter provides the information for the number of outstanding frames allowed. On the sending side, this means that there will never be more than FW outstanding frames. Any other frames will be buffered. On the receiving side, this guarantees that a message from an old frame (more than FW older than the most recent frame) will never be delivered. Setting this parameter to one will provide a stop-and-wait type transmission. A new frame will not be transmitted until the previous frame has been sufficiently received. Setting this parameter to zero allows an infinite number of simultaneous frames.

– Allowable values: non-negative integer.

FD : *frame boundary descriptor*. This parameter determines how the protocol is informed about frame boundaries by having the application mark the first message in a frame or the last message in a frame. Marking the first message in a frame can be used when the *frame size* is known at the time of the transfer of the first message in that frame. Marking the last message in a frame can be used when that information is not known.

– Allowable values: MARK_FIRST, MARK_LAST.

FS : *frame size*. This determines the number of messages in a frame. This information can be used with the *frame transfer percentage* to determine the current percentage of sufficiently received data, as well as to inform the sender and receiver of the number of messages to expect for this frame. This parameter is optional when the *frame transfer percentage* is set to zero and the *frame boundary descriptor* is MARK_LAST.

– Allowable values: positive integer.

Ordering Parameter : ORDERPARAMETERS

MD : *maximum misordering distance*, defined in number of messages. This defines the limit of misordering of messages that the application will allow. If set to zero, no out of order messages are transmitted to the user. If set to some number n , messages that are transferred to the application will be at most n places out of order.

- Allowable values: positive integer.

Loss Tolerance Parameters : (per Data Frame) LOSSPARAMETERS

W : *window size*, defined in number of messages. This represents the sliding window used for counting the number of lost messages. Loss are counted in the most recent W messages.

- Allowable values: positive integer greater than 0.

L : *maximum lost messages in window*. This defines the number of allowable losses within the window W. if L is not satisfied, a retransmission is requested for any messages necessary to satisfy L. Setting this parameter to zero indicates the application tolerates no losses.

- Allowable values: positive integer less than or equal to W

C : *maximum loss burst size*. This defines the maximum number of consecutive lost messages that can be tolerated. Setting this parameter to zero indicates the application tolerates no losses.

- Allowable values: positive integer.

Control Parameters : CONTROLPARAMETERS

SR : *sender/receiver*. This parameter determines which side of the transmission each endpoint is on. Since this specification defines a one way data transfer, there needs to be a distinction between the sender and the receiver endpoints. This parameter defines the direction of the data transfer.

- Allowable values: SENDER, RECEIVER.

Functions :

Initialization : VARHANDLE VARINIT(CONTROLPARAMETERS, FRAMEPARAMETERS, ORDERPARAMETERS, LOSSPARAMETERS). This function sets the appropriate variables for this data transfer. A handle is returned for use with future transactions. During initialization, the protocol performs any necessary functions to set up the connection. Connection establishment and shutdown methods are not addressed here. The handle that is returned represents the endpoint for a communication channel between the sender and the receiver.

Sending Data : VARSEND(VARHANDLE, DATA, FRAMEBOUNDARYPARAMETERS). This function is used by the sending side to transmit data. The parameter FRAMEBOUNDARY informs the protocol that either a frame has ended or that a new frame has started. This is dependent on the value of the *frame boundary descriptor*. At this time, the application can also inform the protocol of the *frame size* if it is a new frame.

Receiving Data : VARRECEIVE(VARHANDLE, BUFFERINFO, FRAMEBOUNDARYMARKER). This function is used to receive data. The delivery of data follows the *frame transfer percentage*. The FRAMEBOUNDARYMARKER indicates to the application that either a frame has ended or that a new frame has started depending on the value of the *frame boundary descriptor*. VARRECEIVE is a blocking call. It returns when there is data that satisfies all guarantees and can be passed to the application. Each call to VARRECEIVE may return multiple messages. If multiple messages are ready to be delivered to the receiver simultaneously, the protocol will try to fill the buffer provided with as many messages as possible. The return value for the function call is the number of messages contained in the buffer. A return value of zero indicates failure. The application may set a maximum on the number of messages to be returned in the BUFFERINFO parameter.

Resetting Parameters : `VARUPDATE(VARHANDLE, FRAMEPARAMETERS, ORDERPARAMETERS, LOSSPARAMETERS)`. Once a communication instance is set up, the control parameters cannot change, but the other parameters can. This change is only allowed on the side of the *control entity*.

5.3 Variable Reliability Protocol

The use of a sliding window mechanism drives the protocol's implementation. In our sliding window protocol, decisions on when to ask for specific retransmissions are made by the receiver. These decisions are made based on the policy defined for the specific protocol. For example, the policy for a reliable protocol would be to ask for a retransmission of all messages determined as lost by the receiver. In such a protocol, an ACK indicates that the message has been received successfully by the receiver. But this definition is a policy decision. For our purposes, we would like to say that an ACK indicates that the sender no longer needs to buffer or retransmit the ACKed message; thus, depending on the policy used in a given protocol, the sender may receive ACKs for messages that were never received. The key is that the receiver controls the ACK policy.

In our variable reliability protocol, the receiver tracks messages using two information sources: the *receive window* and the *receive history*. The *receive window* indicates the range of *active* messages, where an active message is a message that may still be accepted and buffered by the receiver. Information about the reception of messages is maintained in this window in order to prevent the passing of duplicate messages to the application. The *receive history* is used to determine if a loss in the receive window can be allowed; messages in the receive history are no longer active, and they will not be accepted by the receiver. This is because message losses in the receive history have already been "allowed". Additionally, for the remainder of this discussion, the receiver assumes that messages received out of order are lost (i.e., ordering distance is 0). This assumption permits simplification of the protocol at the cost of some unnecessary losses and retransmissions.

The protocol uses cumulative ACKs and NAKs. When the receiver detects a lost message, it checks to see if that loss violates any loss tolerance parameters. If so, the receiver sends a NAK for that message. If not, the message will be acknowledged in the next timer-generated ACK. Due to the fact that the loss of the last message in a window or frame may go unnoticed by the receiver, the sender maintains a timer for each message, which, on expiration, causes the message to be retransmitted. A number of techniques may improve protocol performance for this kind of occurrence, including indicating the number of messages in a data frame or sending markers after the last message in a data frame.

5.3.1 Implementation

5.3.1.1 Protocol State Block

The protocol maintains state for each reliability level that it uses.

- **Sender State Information:** The sender implements a standard sliding window protocol with the following parameters:
 - *MAX_UNACKED*: The maximum number of unacknowledged messages allowed at the same time.
 - *ackExpected*: The lower edge of the sender's sliding window.

- *nextToSend*: The upper edge of the sender’s sliding window.
- *nextToBuffer*: The next sequence number for buffered messages.
- *nOutstanding*: The number of outstanding unacknowledged messages.
- *nBuffered*: The number of currently buffered messages.
- ***sentMsgs*: The list of outstanding messages.
- **buffered*: The list of buffered messages.
- *MSG_TO_TIME*: The time period the sender waits before deciding that a message has been lost.

As a message is sent, the sender buffers it on the *sentMsgs* list and starts a timer. As messages are acknowledged, the buffers are freed and the timers are stopped.

- Receiver State Information: The receiver implements a modified sliding window protocol that can acknowledge messages even if they have been considered lost. The protocol is governed by the following parameters:
 - *holdPoint*: The lower edge of the receiver’s history window.
 - *nextExp*: The sequence number of the next expected message.
 - *tooFar*: The upper edge of the receiver’s reliability window plus one.
 - *waiting*: The lower edge of the receiver’s reliability window and the id of the message that we are waiting for.
 - *arrived*: The bitmap for in bound messages
 - *maxloss*: The maximum number of losses allowed in the receiver’s reliability window.
 - *maxconsloss*: The maximum number of allowed consecutive losses.
 - *losscnt*: The current number of losses in the receiver’s reliability window.
 - *conslosscnt*: The current number of consecutive losses.
 - *ACK_TO_TIME*: The time period the receiver waits before deciding that something has gotten lost.

The modification to the receiving side of the sliding window protocol affects the allowance of lost messages. The relationship between the parameters *holdPoint*, *waiting*, *nextExp* and *tooFar* determine the action to be taken upon the receipt of a message. The protocol defines two windows: the history window and the reliability window (see Figure 25). The history window maintains information about the history of allowed losses. The reliability window maintains information about current messages and losses. The history window is determined by the values of *holdPoint* and *waiting*. The size window will always be less than the reliability window size. The reliability window is determined by the value of *waiting* and is always the same size (*MAX_VAR_UNACKED*). *nextExp* always falls somewhere in the reliability window.

Pseudocode for an implementation of this protocol can be found in Appendix D.

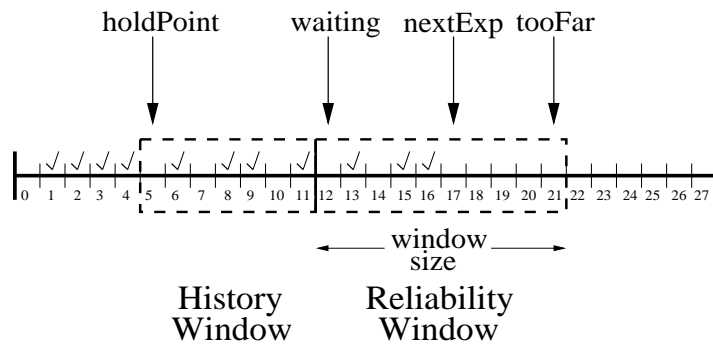


Figure 25: Example of History Window and Reliability Window

5.3.1.2 Example

The following figures show an example of the receive window and the receive history during execution. In this example, the receive window size is 10 and at most 3 messages may be lost out of any 10 consecutive messages. The maximum number of consecutive losses is 1. The variable *nextExp* is the sequence number following the highest-numbered received message. The variables *waitPoint* and *tooFar* delimit the receive window. The value of *waitPoint* indicates the message that the receiver is currently waiting for. If the receiver is not waiting for any outstanding messages, then *waitPoint* is set to *nextExp*. *tooFar* is always *waitPoint* plus the receive window size. In the case of an unacceptable loss, *waitPoint* will be set to the sequence number of that message. The variable *holdPoint* indicates the oldest lost message in the receive history. This and any other later allowed losses in the receive history will determine if a new loss can be allowed. *holdPoint* is at least *waitPoint* minus the receive window size.

The example begins with *holdPoint*, *waiting* and *nextExp* equal to 0, and *too far* equal to 10 (*waiting* plus the window size) (Figure 26). As data arrives (message 1), *waiting* and *nextExp* are

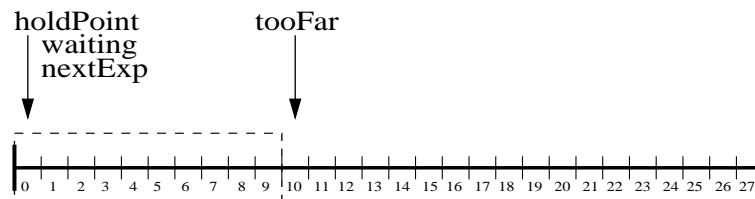


Figure 26: Example of Variable Reliability Protocol – Step 1

advanced to 2 and *holdPoint* stays at 0 (Figure 27). This is because the receiver noticed that message 0 was lost. This loss does not violate any of the loss requirements so the the loss is allowed. We need to maintain that information about this loss in the history window until it can no longer affect the messages in the reliability window. The next three messages (2, 3 and 4) arrive successfully and then message 6 arrives (Figure 28). This indicates that message 5 was lost. This does not violate the loss parameters, so *waiting* and *nextExp* are incremented to 7. This continues with the arrival of message 8, noting the loss of message 7 (Figure 29). With the arrival of message 9, the loss of message 0 no longer affects the reliability window, and so the history window can be advanced by jumping *holdPoint* to 5 (Figure 30). Since messages 1 through 4 were received

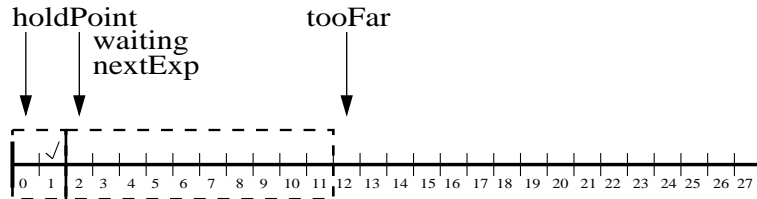


Figure 27: Example of Variable Reliability Protocol – Step 2

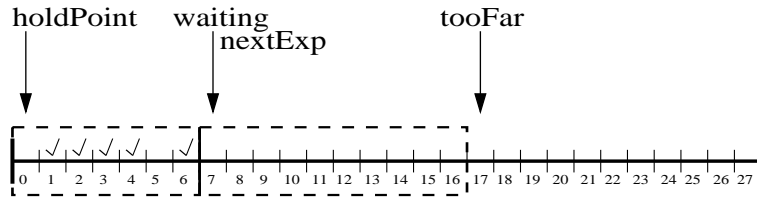


Figure 28: Example of Variable Reliability Protocol – Step 3

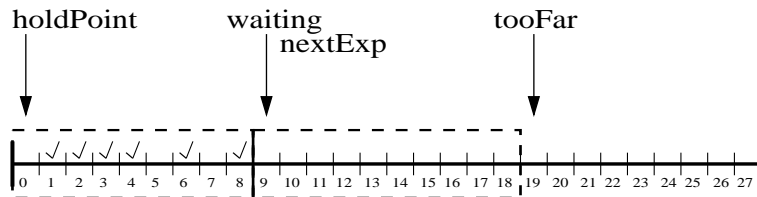


Figure 29: Example of Variable Reliability Protocol – Step 4

correctly, we need not keep them in the history window. The arrival of message 11 indicates the

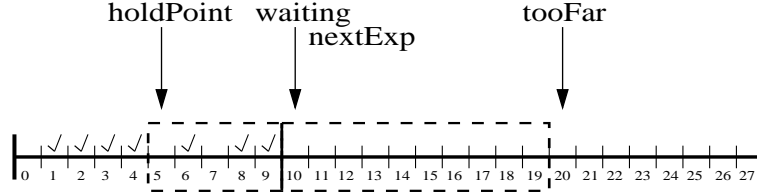


Figure 30: Example of Variable Reliability Protocol – Step 5

loss of message 10 (Figure 31). There are now three losses in the history window. The arrival of

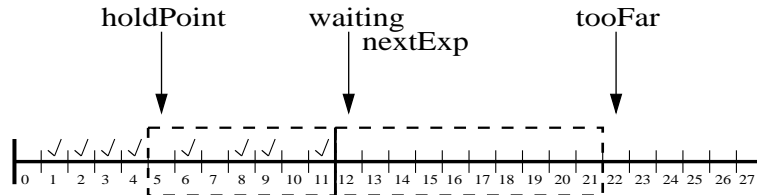


Figure 31: Example of Variable Reliability Protocol – Step 6

message 13 indicates the loss of message 12 (Figure 32). This additional loss violates the maximum number of losses in window parameter. We now ask for a retransmission of message 12. At this point, *waiting* stays at 12, indicating that 12 is the message that we are waiting for, and *nextExp* advances to 14. Messages 15 and 16 arrive before the requested retransmission of 12 (Figure 33).

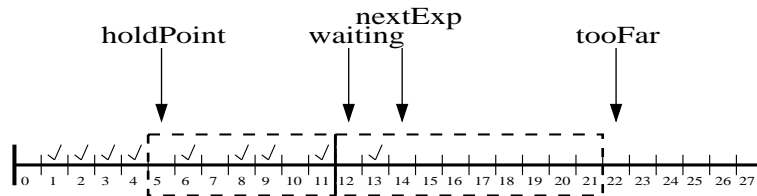


Figure 32: Example of Variable Reliability Protocol – Step 7

nextExp is advanced, but *waiting* still stays at 12. When the retransmission of message 12 arrives,

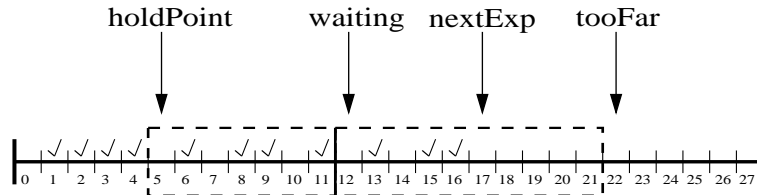


Figure 33: Example of Variable Reliability Protocol – Step 8

waiting is advanced to 14, since the loss of 14 violates the same loss parameter (Figure 34). A

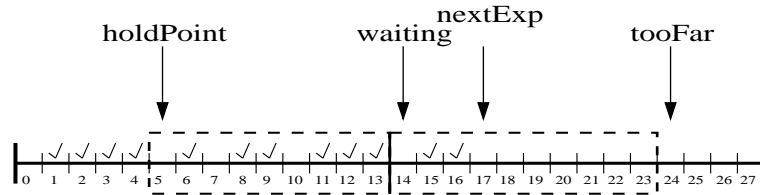


Figure 34: Example of Variable Reliability Protocol – Step 9

retransmission is requested for 14. Message 17 arrives before the retransmission of message 14, and so *nextExp* is advanced, but *waiting* is not (Figure 35). The arrival of message 14 allows us to

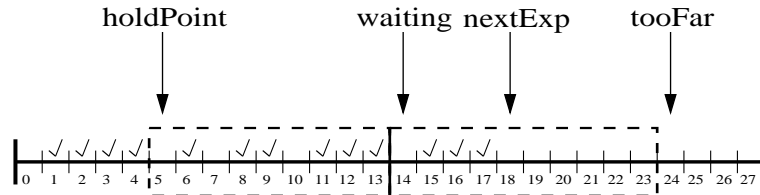


Figure 35: Example of Variable Reliability Protocol – Step 10

advance *waiting* to 18, and to advance *holdPoint* to 10 (Figure 36).

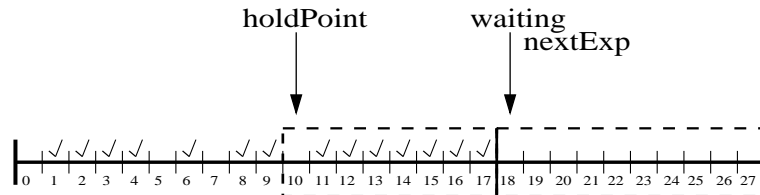


Figure 36: Example of Variable Reliability Protocol – Step 11

5.3.2 Experimental Results

These results were gathered for the transmission of a large image using the variable reliability protocol running under varying network conditions. The experiments were run to validate the implementation of the variable reliability protocol. For these experiments, a sample represents the average of ten runs with those specific parameters (reliability level and average loss rate). Loss rates varied from 0-0.33%, and reliability was varied from 100-0%. Figure 37 shows the effect on transmission time. As can be expected, the higher reliability requires more time for situations where many messages are lost or corrupted. For lower reliability levels, the reliability mechanisms do not react until higher loss rates are seen. This can be seen in the fact that the 33% reliability stream is not affected until the higher loss rates are achieved.

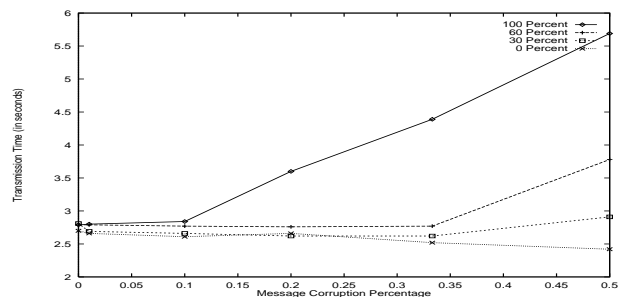


Figure 37: Transmission Time

5.4 Overview of Contributions

Our research contribution is a protocol for implementing variable reliability communication. The use of this protocol provides the communication layer with opportunities to realize performance gains by exploiting tradeoffs with respect to the reliability of message communication. Specifically, with this protocol, the communication layer can express the application’s data reliability requirements more precisely. This information can then be combined with feedback from the network resources being used to adjust communication protocol parameters during execution, thereby improving use of available network resources.

Specifically, we have presented a mechanism with which application data reliability can be precisely specified. We have implemented an infrastructure that allows one application data stream to be segmented into multiple reliability streams. Each of these reliability streams can have its own specification of reliability, independently of the the other streams.

Chapter 6

Payoff-Based Adaptation

In the previous chapter, we demonstrated that explicit control of reliability by an application can lead to improved performance. In this chapter, we demonstrate the effects of adapting communications on behalf of the application based on information supplied through a “payoff-function”-based interface between the communication layer and the application. Additionally, we demonstrate the effects using more detailed network resource information in the form of loss-load curves. Payoff-based adaptation uses the concepts described in Chapter 2 to optimize the value of channel service to the application. In this chapter, we explore the effects of optimizing the service of one data stream by adapting two communication parameters: reliability and transmission time. Configuration decisions are based on information local to that data stream.

6.1 Target Application

Image transfer is the basis for many multimedia applications and is often tolerant to some loss. This tolerance makes it a suitable candidate for taking advantage of adaptable communication. Turner and Peterson describe a retransmission-free protocol for image transfer[66]. They make the assumption that applications will be willing to give up some degree of image quality for the decrease in delay. The application is given the choice of receiving all or none of the retransmissions. We believe that an application like this can benefit from the ability to make more precise tradeoffs between delay and reliability.

One specific application driving our research is a distributed virtual environment operating across the Internet. The environment was designed as a collaboration tool that supports concurrent interactions among multiple users. The environment permits users to navigate through a virtual world organized as multiple rooms. The virtual world as well as the objects in the world are represented using graphical, audio, video, and data visualization techniques. Users interact with objects in the world, with the world itself, and with the other users in the world. Rooms are separated by walls, but there may also be doors and windows present that allow users to see parts of other rooms or hear pieces of conversations being conducted elsewhere. Objects themselves may be stationary, movable, or they may even move on their own (i.e. other users or autonomous agents). Objects may be simple or complex graphical representations, still or motion images, or data visualizations, and may have associated audio.

The dynamic nature of the virtual world coupled with runtime changes in users’ interests result in frequent and dynamic changes in users’ service requirements. For example, while a user is relatively “far” from a video monitor visible in the environment, there is no need to offer high resolution real-time images for display on this monitor. However, both display rate and image quality must be improved to realistically model the fact that the user can see the image more clearly when approaching the monitor. In [51], the idea of *Importance of Presence* is introduced

to help determine the service requirements of objects in a distributed virtual environment. Due to the fact that each user in the virtual world may have different interest levels regarding different objects in the world, it is not beneficial to multicast all of the data to all of the users. One suggested solution to this problem is the use of layered multicast techniques [2], where data is split into multiple layers and receivers can determine what layers they want to receive. Our approach is to allow receivers to specify two additional pieces of information. First, individual receivers can specify different layerings of the data. This can correspond to different focus areas in the transmitted data. Second, we allow receiver-specific specification of communication protocols, as well as the configuration of the specific protocols, to be used for each such layer. For example, different receivers may receive the same data at different reliability levels or transmission rates. We take this approach due to the expected diversity of user's interests and requirements.

The communications in our distributed virtual environment fall into three categories. The first category includes data that must be transmitted reliably, including control messages and the one time transfer of some object descriptions. This type of data is very sensitive to packet loss and its transfer reliability must not be sacrificed. The second category is composed of data streams, where data must be segmented into multiple messages in order to be transmitted. Examples include changing texture maps, sensor inputs or continuously generated scientific data. This type of data often offers some level of redundancy. At the application level, it may be possible to tolerate data loss or compensate for it by inferring the lost data from other messages that are "spatially" close to it. For example, a lost message in an image can often be compensated for by examining the image surrounding the lost message and performing some type of averaging function. Such recovery has limitations in the number of "nearby" or consecutive data losses it can tolerate. The third category consists of continuous streams of relatively small messages, each of which is an update to some specific application data. Examples include object position updates or tracker updates, which have the property that, if a message is lost, the next message will provide the newest information for the data [10]. Due to this property, there is little tolerance for waiting for retransmissions, since much of the lost data can be approximated from the next message much faster than waiting for the retransmission. Although continuously updated, this type of data is very sensitive to loss due to the effect on the lag perceived by the user.

Some types of data cover multiple categories. One specific example is real-time video. Each video frame is a large data set and needs to be segmented into multiple messages. New frames are continually being transmitted and the loss of some data from one frame may be tolerated due to the fact that the next frame will provide an updated image. Real-time video transmission also introduces many issues involving timing constraints that we do not address in this section. Instead, this section focuses on issues involving the transfer of large data sets that may have soft timing issues, but are not time critical. This aspect of data transfer is still applicable when considering the transfer of the individual frames of a video stream. Our distributed virtual environment currently supports video objects, where images are captured from a video camera in real time and transmitted to interested users. We are currently investigating the effect of using our payoff-based techniques to determine appropriate update rates for the video streams in our distributed virtual environment.

6.2 Adaptive Quality Specification

Data manipulated by applications can often be grouped into different levels of quality. Quality may refer to running time, reliability or any other similar service parameters. In any case, the meaning of "quality" is specific to each application, and quality values may be equated to application data

at many levels of granularity, ranging from data sets to frames to individual messages.

We use two distinct examples to illustrate these concepts of adaptive quality specification and its interaction with our variable reliability mechanism. The first example explores static quality specification with an image transfer application that requires different reliability levels for different fixed regions of the image. The second example explores dynamic quality specification in a distributed virtual environment that dynamically changes the reliability requirements of regions of its world. For both of these applications, simple payoff functions are used. Payoff functions for quality correspond to reliability levels in the variable reliability protocol. As reliability requirements change, the application changes the payoff function for that data. Each payoff function has a maximum payoff at the respective quality level, and is zero below that level. These payoff functions cause the communication layer to transfer data at the reliability level that provides the appropriate quality level.

6.2.1 Locality-Based Adaptive Quality Specification

Image transfer provides an excellent avenue for statically partitioning data into multiple quality regions. In image transfer, each image is a data frame. If we consider the transfer of an image of a person, the most important part of that image might be the face (see Figure 38(a)). This section of the image must be of high quality, while the quality of the rest of the image is allowed to decrease as we move further away from the face. This may be expressed by specification of a separate payoff function for each section.

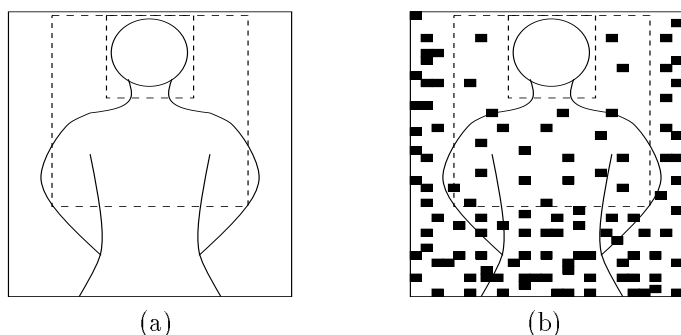


Figure 38: Image with Application-Specified Quality Regions

We can now combine the concept of the variable reliability protocol with quality-based data partitioning. Through this combination, we can ensure that the face is received at 100% reliability, while the reliability levels of the rest of the image decrease as we move away from the face. (see Figure 38(b), black spaces indicate lost messages.)

For our locality-based experiments, we use an application that transfers a 6.5Mbyte image across a 10Mb Ethernet connection. The application runs under three different partitioning configurations. Configuration 1 imposes 100% reliability on the entire image. Configuration 2 partitions the image into two regions: 11% of the image (e.g., the face) requires 100% reliability; 89% requires 66% reliability. Configuration 3 partitions the image into three regions: 11% requires 100% reliability; 28% requires 66% reliability; and 61% requires 33% reliability. Each of these configurations is run at progressively higher message loss probabilities. We simulate random loss of messages.

Figure 39 depicts the running times for the transfer of a partitioned image in a lossy network under each of these conditions. The results are an average of 10 runs at each loss probability

rate. The closer we come to specifying the reliability requirements of the application, the better we can judge the necessity for message retransmission. As the loss increases, the running time for configuration 1 increases to 280% of the running time for a run with no losses. In comparison, configuration 2’s running time increases to 180% and configuration 3’s running time to only 136%. The improved transmission time in configuration 3 is gained while remaining within the reliability requirements of the application.

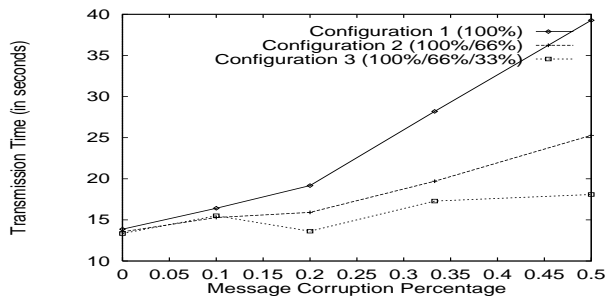


Figure 39: Transfer Time for Quality-Based Partitioned Image

6.2.2 Distance-Based Adaptive Quality Specification

The second application used for evaluation is a simulation of a distributed virtual environment, where multiple users operate in a joint world and share a world view. The world view in this application is processed as an image and passed back and forth between the users as it is being updated. The world is decomposed into blocks, where each block has an owner that is responsible for coordinating reads and writes as well as sending updates to the other users. As each user moves through the world, it updates the data in the world and receives updates from other users.

The quality-based partitioning of data and the variable reliability protocol lend themselves well to being used for this application, since each piece of data potentially has a different reliability requirement, and these requirements may change as the simulation evolves. Intuitively, each user only cares about the immediate surroundings. In other words, the reliability of an update is determined by the update’s proximity to the user receiving the update. The sections that are out of the user’s view may only require periodic unreliable updates, or no updates at all. The application can dynamically change the assignment of data to a specific reliability level. This allows the application to choose what data is most important to it, and pay the overhead of reliability for that data.

For this application, the world is divided into concentric “rings” around the location of the user. Figure 40 shows an example from the viewpoint of user 0. Each of these rings corresponds to a reliability level. The loss tolerances in the distant rings are higher than those of the closer rings. In other words, as the updates come from “further away”, their reliability is relaxed.

The goal of the distance-based experiment is to determine the performance impact of allowing users to change the reliability of updates. Experiments monitor the number of retransmissions required and the number of allowed losses. Results are computed based on the number of updates received and the number of messages retransmitted. These numbers are averaged over four runs. Table 5 shows results for user 0 using a configuration with 100% reliability and using a configuration with the type of partitioning described above. In the runs of the reliable world, all losses trigger retransmission requests. In the runs using quality-based partitioning, only 7-22% of the losses

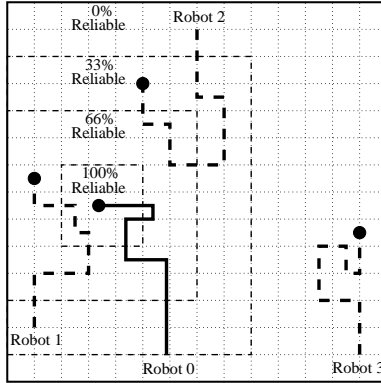


Figure 40: Snapshot of Distributed Virtual Environment Application with Variable Reliability Rings

trigger retransmission requests. This percentage increases as the degree of requested reliability is increased. For a more detailed explanation of this experiment and the results, see [39].

	Average Number of Updates Received	Average Number of Messages Retransmitted
100% Reliability	15377.00	5058.25
Quality-Based Reliability	11838.00	1728.00

Table 5: Average Updates Received and Messages Retransmitted (User 0)

6.2.3 Initial Conclusions

With these experiments, we have demonstrated that, by providing detailed information about the quality requirements of the application, the performance of the application can be improved. The cost of such improvements is decreased quality, within an acceptable range, for specified data. In both experiments, the quality specifications of the applications are used to determine the communication configuration that best fits these specifications. The second experiment demonstrates that, by adapting to dynamic changes in application specifications, the communication configuration can be adapted to continue to provide the application with acceptable service. Next, we will focus on changes in network resources and their effects on communication configuration and channel service.

6.3 Payoff Adaptation with Observed Network Behavior

This section describes the results of two different experiments designed to explore the effects of end-to-end adaptation. Both experiments use an application that transfers a 6.5Mbyte image

across a 10Mb Ethernet connection. This is a simplified version of the image transfer application described in Section 6.2. In these experiments, the entire image is set to be at one reliability level.

For both experiments, the sender determines a course of action using local information and information collected by the receiver. Specifically, in response to changes in the error rate, the sender can change the reliability level being used in order to achieve an acceptable transfer time. In this experiment, the sender can dynamically switch between four distinct reliability streams: 100% reliability allows no losses; 66% reliability allows 1 consecutive loss and 34% loss in a window; 33% reliability allows 3 consecutive losses and 67% loss in a window; and 0% reliability allows any loss it notices.

In order to determine the benefits of using payoff adaptation, we compare it to the results of using simpler, threshold-based adaptation and of using fixed reliability levels. The sender using payoff adaptation employs the techniques described in Section 2.4. For threshold adaptation, the sender uses a fixed scheme for placing values on resource information. Specifically, whenever a predefined threshold is crossed, the sender automatically changes the communication to a lower or higher reliability level. In this example, the sender monitors the message loss, which is quantized into four ranges. Each range is associated with a reliability level, as specified above. If the observed message loss crosses a threshold from one range to the next, then the sender adjusts to the reliability associated with the new range. In our application, if the sender has been experiencing 1% message loss, which then jumps to 10%, the sender would reduce the reliability level from 100-66%. The placement of the thresholds within the range of message loss determines the performance of the application.

For this application, two service parameters are specified: ObservedReliability and TransferTime, as defined in Section 2.4. For this experiment, we focus on the effect of adapting the reliability level and leave the transmission rate fixed at 650KBps. The operating points for the functions for observed reliability and transfer time, F_{or} and F_{tt} , were collected prior to the final experiment using experiments similar to those discussed in Section 5.3.2. Total payoff is defined as $P_{total} = P^{or}(ObservedReliability) * P^{tt}(TransferTime)$. P_{total} is evaluated over the available reliability levels, given the currently observed loss rate. By using the steps described in Section 2.4, the communication layer can determine the reliability level that maximizes total payoff for the application. The function for determining total payoff for this experiment was chosen to be multiplication. Multiplication retains the payoff scale of 0-1 and implies that the total payoff will be zero if any of the individual payoffs are zero.

The payoff functions that we use are shown in Figures 41 and 42. These payoff functions represent some features matching the application’s behavior; in particular, the reliability drops off at a relatively slow rate up to 50% reliability and drops quickly after that. The payoff function for time allows for some minimal delay at a high payoff, but drops off sharply after twice the expected transfer time.

Figure 43 shows the final payoff for each of the adaptation algorithms as well as for two fixed reliability levels. The goal of the payoff adaptation method is to adaptively choose the best running point for the application. We can see by the final payoff values in Figure 43 that the curve for the payoff adaptation method is essentially an upper envelope for the other curves and outperforms them at various points. This illustrates two points. First, a simple adaptive algorithm can exploit the benefits provided by the variable reliability protocol infrastructure, and it can choose the correct operating points efficiently. In other words, it can correctly choose the reliability level at which to run. Second, end-to-end adaptation has the potential to provide the communication layer with sufficient feedback to adjust reliability.

It is illustrative to look at the raw running times of adaptation algorithms and to compare those times to their effectiveness in terms of the total number of bytes transferred. Figure 44

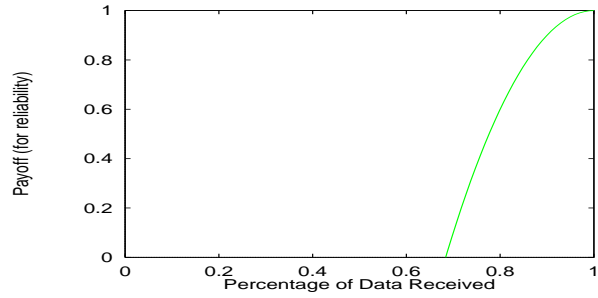


Figure 41: Payoff Function for Reliability

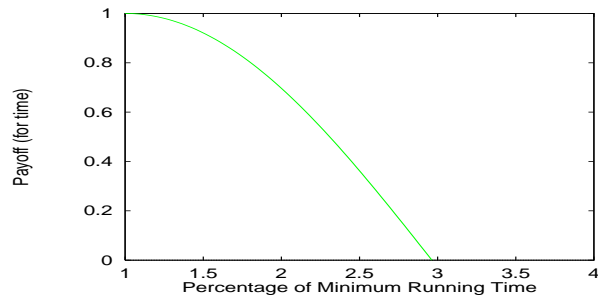


Figure 42: Payoff Function for Time

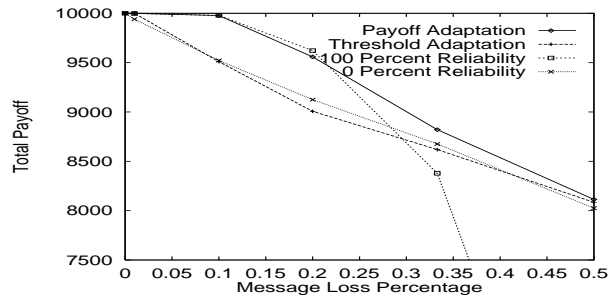


Figure 43: Final Payoff

shows the running times for the two adaptation methods and two fixed reliability runs. The results in Figure 44 show that for most of the experiment, the payoff adaptation method runs slower than the threshold adaptation method. If we consider the total amount of data that is received, we can see in Figure 45 that the payoff adaptation method also receives more of the transmitted data than the threshold adaptation method. These results should be expected, since the payoff adaptation method adapts to optimize payoff for both the amount of data received and the running time.

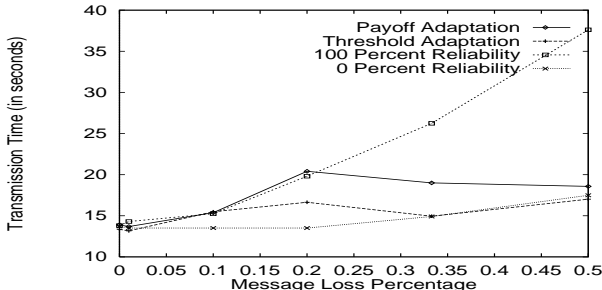


Figure 44: Time Statistics

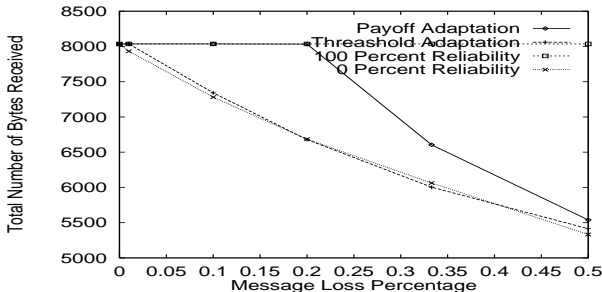


Figure 45: Data Statistics

The results in this section demonstrate that communication configuration based purely on application requirements does not provide efficient service to the application. On the other hand, given some knowledge of the current behavior of the network, the communication layer can adapt the configuration to effect more suitable service for the application. Additionally, combining application specification and network resource information, the communication layer can maximize the payoff to the application for that service.

6.4 Payoff Adaptation with Loss-Load Curves

This section addresses the use of service availability curves in conjunction with payoff adaptation. Network resource information is represented in a fashion similar to the loss-load curves described in Section 2.3. By supplying the communication layer with a specific loss rate for each transmission rate, the communication layer can use this information to determine expected transfer time experienced at each rate. Note that in this example, transmission rates are no longer fixed. Additionally, since we are using loss-load curves, the loss rate is implied from the specific transmission

rate. The operating points for the functions for observed reliability and transfer time, F_{or} and F_{tt} , were determined from baseline timing runs. Data points were gathered for reliability levels of 70, 80, 90 and 100%; transmission rates of 400KBps, 500KBps, 600KBps, 700KBps and 800KBps. Each run is subjected to loss probabilities varying from 0-50% in increments of 5.

From loss-load curves in the form $F_{LLCurve}(\text{TransmissionRate}) = \text{LossRate}$, the communication layer can determine the estimated transfer time for each transmission rate and reliability level. Next, transfer time and end-to-end loss are used as parameters to the application-supplied payoff functions P_{OR}^{rel} and P_{TT}^{time} . Finally, by evaluating the payoff functions (Figure 46 and Figure 3) to maximize P_{total} , the communication layer can determine the transmission rate/reliability pair that it considers the best fit for the application’s requirements. The payoff function for reliability is shown in Figure 46, while the payoff function for time is the one shown in Figure 3. Adaptation is performed in response to changes in network resource availability, as specified by changing loss-load curves. As the communication layer is informed about a change in the loss-load curve, it reevaluates the payoff functions to determine whether it should reconfigure the communication parameters.

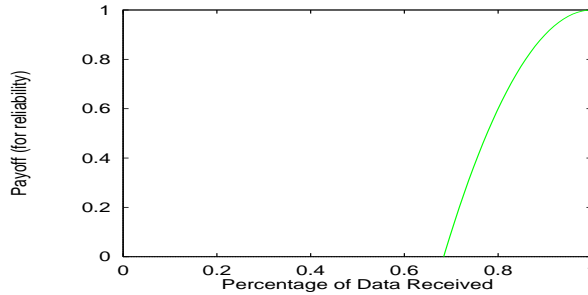


Figure 46: Payoff Function for Reliability

For a specific example, consider Figure 4 which shows a loss-load curve with simultaneous increases in loss and transmission rates. Figure 47 depicts the payoff values obtained for the loss-load curve in Figure 4 over varying transmission rates. The net payoff is calculated as the product of P^{rel} and P^{time} . Each line in Figure 47 represents the transfer of the image at the given reliability for the set of transmission rates. Using these curves, we can determine the optimal transmission rate and reliability for this specific loss-load curve and the requirements of the application by finding a maximum for P_{total} . In this example, the communication layer should choose to transfer data at a speed of 600KBps and a reliability level of 90%.

For these experiments, we use an application that transfers a 6.5Mbyte image 30 times across a 10Mb Ethernet connection, where the communication layer monitors the loss-load curves supplied by the network. The communication layer has the ability to change two parameters: reliability and transmission rate. For reliability, the communication layer can change the acceptable loss-burst size and allowable loss percentage in a window. In theory, the communication layer could allow a continuous choice for reliability. Our current implementation permits the choice between four distinct reliability levels: 100, 90, 80 and 70%. For the transmission rate, rates from 400KBps to 800KBps in increments of 100 are chosen.

The experimental results in this section demonstrate the effects of changes in loss-load curves over time, which model changes in network service (e.g., the network becoming more and less congested). The loss-load curves used are based on the one in Figure 4. The linear loss-load curve in Figure 4 imposes a 5% increase in loss for each 100KBps increase in requested bandwidth.

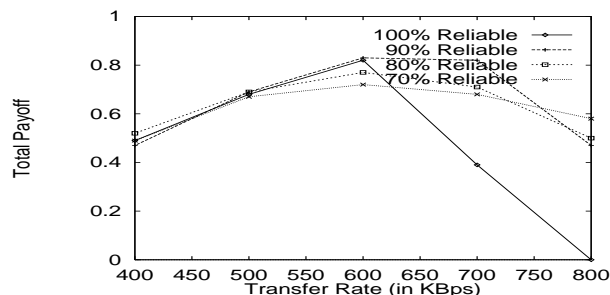


Figure 47: Net Payoff Curves

Changes in the loss-load curve are represented by shifting the entire curve up or down along the y-axis, thus imposing more or less loss depending on the way in which it is shifted. In this experiment, the loss-load curve is shifted every 15 seconds.

The experiment compares the results of two runs of the application. In the first run, the communication is static. The application transmits at 800KBps and requires 100% reliability. In the second run, the communication is dynamically controlled through payoff-based adaptation. The communication layer monitors the loss-load curves and recalculates the communication parameters as the loss-load curves change. The solid line in Figure 48 represents the loss over time seen by an application that is transmitting at 800KBps. The dashed lines in Figure 49 show the dynamic choices made by the communication layer as to reliability and offered load. Due to the choice of offered load, the loss rate experienced over time is represented by the dashed line in Figure 48. These decisions are based on the payoff calculations made by the communication layer at each change in the loss-load curves. The total payoff for both runs at each change is shown in Figure 50. The solid line represents the total payoff for the first run and the dashed line represents the total payoff for the second run.

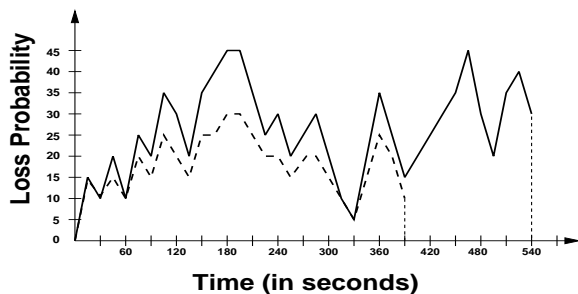


Figure 48: Observed Loss over Time

Figure 50 shows that, given the ability to choose communications parameters, the application's payoff can be maximized at discrete points in time. In addition, final payoff is shown in Table 6, which depicts the total amounts of data received and the total transfer times for both runs of the application. As expected, the static run receives the maximum payoff for the amount of data received. In comparison, the adaptive run has relatively high payoff for the amount of data received. The biggest difference is in the payoff for the transfer time. The product of these two payoff values gives us the total payoff for the transfer. The static run pays off at 0.8165, while the adaptive run

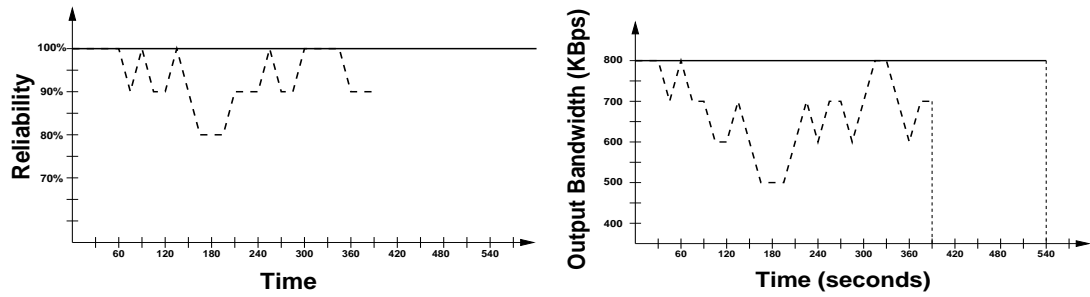


Figure 49: Application Reliability and Bandwidth over Time

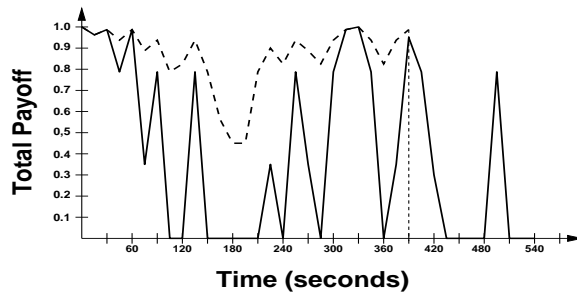


Figure 50: Total Payoff over Time

pays off at 0.9627. As expected, the application that adapts to fluctuations in network resources is able to make intelligent decisions based on its willingness to trade off reliability for transfer time.

	Data Received	Fraction of Total Data Sent	Payoff for Data	Transfer Time	Percent of Minimum Time	Payoff for Time
Static	109350 msgs	1.00	1.000	537 sec	177	0.8165
Adaptive	104874 msgs	0.959	0.9832	381 sec	126	0.9784

Table 6: Results for Amount of Data Received and Transfer Time

The experiments in this section demonstrate that service availability curves, in the form of loss-load curves, can provide useful information when determining an acceptable communication configuration. By using the information provided in the loss-load curves, the communication layer can determine the effect of a range of transmission rates on the loss rate seen by the application. This foreknowledge provides the communication layer with more fine-grained information than it could get from observational techniques, and so allows the communication layer to make more informed decisions during adaptation.

6.5 Overview of Contributions

Distributed applications operating in dynamic network environments may experience unpredictable changes in resource availability. Our intent is to enable such applications to operate within acceptable parameters despite potentially insufficient network resources. Our solution approach is the provision of an adaptive communication layer that configures communication protocols based on the benefit the communication layer expects to realize for the application. To this end, we have presented our communication layer which provides a mapping between the services available from the network and the services needed by the application. Through this mapping, the communication layer monitors and adapts the service provided to the application using payoff-based adaptation techniques. These techniques use knowledge of application service requirements and network service availability to maximize the perceived benefit to the application.

In this chapter, we demonstrate that, with information about the requirements of the application, the communication layer can improve the match of application requirements with network resources. Similarly, with information about the state of the network, the communication layer can compensate for such changes. Additionally, our work provides a cooperative solution in which the communication layer uses both application resource requirements and information about network resource availability to determine how to configure the communications. Through experimentation, we show that payoff-based adaptation using both types of information can provide communication services that are better suited to applications in situations where both application requirements and network resources are dynamically changing.

Chapter 7

Power-Aware Communication for Mobile Computing

The demand for low-cost, power-efficient computing platforms dominates the mobile computing community, with machines ranging from high-end laptops to small hand-held devices. Yet, these power-efficient mobile computers must support an ever-increasing range of communication services, including Internet access, multimedia conferencing and collaborative work. In part, this trend is driven by the fact that wireless communication devices are becoming increasingly common, sometimes even replacing traditional Ethernet cards in mobile computers. Unfortunately, as communication demands increase, so do the demands on the machine's batteries.

The overall goal of power management for mobile computers is to prolong battery life by controlling the energy consumption of the mobile host's various devices. Past research has developed techniques to reduce the energy consumption of individual devices, including disks [25, 30, 44], CPUs [28, 45, 67], and network interfaces [65, 17, 49, 41]. All of these techniques aim to reduce the energy consumption of individual devices on the mobile host. This chapter builds on such work. It differs from it, however, by not only attempting to minimize the energy consumption of each device, but instead, by doing so in the context of the total energy consumed by the mobile host.

The specific device investigated in our work is the host's communication device. While we also wish to reduce this device's energy usage, our actual goal is to minimize the total energy consumption of a mobile host in the context of its communication. We approach this problem by defining a *communication action* as the entire set of steps or computations involved in performing some communication via this device. We define the energy consumption of a communication action to be the energy consumed by the mobile host plus the energy consumed by the devices being used during the lifetime of the communication action. We then consider the effects of certain device-level power management techniques on the energy consumed by entire communication actions. This chapter demonstrates the importance of action-based vs. device-level power management. For instance, by reducing the energy consumption of the communication device, total transfer time may be increased, with the resulting effect of increasing the total energy consumption of the communication action. Clearly, this is not a desirable outcome.

Our research presents techniques for power-aware communication for mobile computers. Central to our work is the aforementioned notion of communication actions, for which we capture both the energy consumed for data transmission by the network interface device (e.g., wireless Ethernet, packet radio modem) and the energy consumed by the host when it generates or consumes this data. Based on this information, we determine the energy cost of transmitting data. Next, we control communication actions' energy consumption by adapting certain parameters of the transmission protocols being used, in response to information about the current state of the network (e.g., loss rate, loss burst size).

Consider the action of transmitting a data object using a stop-and-wait transmission protocol. This protocol attempts to minimize the number of messages transmitted, thereby optimizing the energy consumption of the communication device for the entire data transfer. Alternatively, when using a go-back-N transmission protocol, the communication action may incur additional overhead resulting from unnecessary retransmission of successfully received messages, but it also tends to reduce the action's overall transfer (i.e., completion) time. The integrated power management techniques presented in this chapter are designed to consider and capitalize on such tradeoffs.

We demonstrate the effectiveness of power-aware communication techniques with a transmission protocol tailored for wireless communication. In this context, it is intuitive that excessive retransmission and acknowledgments will consume additional energy, because the transmission of these extra messages increases the energy consumption of the wireless network interface device. Recent research in this area has fine tuned transport and MAC layer protocols to minimize this overhead [17, 64, 72]. However, the resulting reduction in the number of unnecessary messages is achieved at the cost of an increase in the total transfer time of the data being sent. In other words, for the communication actions implemented by these protocols, the energy consumed by the network interface device is reduced, whereas the energy consumed by the mobile host's CPU and the other devices involved in these actions is increased.

Consider the design of a transmission protocol. The basic mechanisms used to build this protocol concern/affect window size, acknowledgments (ACKs), FEC and timers. Window size is adjusted to compensate for congestion and to manage flow control. Acknowledgments are used to indicate the state of the receiver. FEC is used to compensate for expected losses, and timers are used to determine certain types of losses. The manner in which these mechanisms are employed defines the behavior of the protocol and therefore determines the energy consumed by the transfer of the data. This chapter explores the use of a selective acknowledgment-based (SACK) transmission protocol with two modes of transmission. The first mode is a standard SACK protocol [48, 37, 26], while the second mode uses multiple retransmissions in the face of lost messages. The principal experimental results attained with this protocol show that under specific (lossy) wireless network conditions, aggressive retransmission policies can reduce the total energy consumption of the mobile host. Specifically, as losses approach 25%, our techniques result in a 25% energy savings for a high-end laptop, and the predict a 20% savings for a mid-level laptop.

The final piece of this research proposes the runtime adaptation of communication parameters in order to minimize the energy consumed during active data transfer. Information about the network environment and the energy consumption of various transmission configurations is used to drive such adaptations. Our goal is to compensate for fluctuations in energy consumption due to the dynamic nature of the services available from wireless communication devices.

In Section 7.1, we describe our power management approach and techniques, and we describe how to quantify and optimize energy consumption. We then discuss power management in the context of transmission protocols in Section 7.2. Section 7.3 presents the results from our experiments with a configurable SACK-based protocol. In Section 7.4, our power management techniques are placed into the context of prior work on communication adaptation. Finally, Section 7.5 presents some conclusions and future work.

7.1 Mobility, Communication and Power Management

The desire to conserve energy during active communication has driven diverse power management techniques. The goal of such power management is simply to conserve battery lifetime for the mobile host. By aiming our efforts at the whole machine, we can consider all of the factors we

have discussed so far. Interestingly, for ongoing communications, it is less important to distinguish the contribution to total energy consumption from each device than to understand the tradeoffs in terms of total energy consumption for different protocol configurations for such communications. In order to be able to quantify such tradeoffs, we present an energy model based on the energy components of a communication action.

This section first discusses the target mobile environment for our research. Next, we discuss in detail how energy consumption may be described to enable online power management for communications. Three sets of measurements are necessary to differentiate base energy usage at idle times, with and without attached network interface devices, from energy requirements during ongoing communications.

7.1.1 The Mobile Environment

Consider a mobile host that communicates with a stationary base station. Power management techniques must deal with situations in which the mobile host is mostly sending data, receiving data, or some combination of both. Specifically, a sender may be concerned with the energy overhead of transmitting unnecessary retransmissions, while a receiver may be concerned with unnecessary acknowledgments. Both sender and receiver are affected by the total amount of time of the data transfer. We discuss both sender and receiver scenarios in the experiments discussed in Section 7.3.

The use of wireless links poses a number of unique problems, including novel loss characteristics, synchronization of disconnected operations, and issues involving packet forwarding. These problems pose significant challenges for end-to-end communication protocols. Two types of models have been studied [5]. The first model exploits the natural hop between a base station and the mobile host for communications crossing the wired-wireless boundary. Namely, standard communication protocols are used by wired hosts to a base station and specialized protocols are used for the final hop from the base station to the mobile hosts [4]. The second model utilizes and tunes existing end-to-end protocols, providing help and hints along the way [6]. We focus on the first model of communication, which allows us to isolate and target the communication between the base station and the mobile host.

7.1.2 Energy Consumption Model

Previous work has focused on the energy consumption of individual devices, such as network interface devices [17, 64, 72]. We consider an entire communication action, thereby addressing both the network interface and the rest of the mobile host involved in the communication. In effect, we are “charging” the communication action for all system usage that occurs during that action. This chapter employs a simple, additive model for the energy consumption of a communication action which ignores concurrency between actions. This simplification is appropriate for mobile hosts not capable of concurrency and also where communication delays make the use of concurrency impractical.

From Figure 12 (Section 4.4.1), it is apparent that transmission-time energy consumption can be partitioned into three sources: communication-specific, device-specific and machine-specific. The machine-specific and device-specific components are only affected by the total amount of time required for communication. On the other hand, the communication-specific component, which represents the amount of additional energy needed for actual data transmission, is affected by two parameters. The first is the amount of energy consumed by the device for data transmission (or data reception for a receiver). This amount will generally increase linearly with data size. In a

lossy network environment, this component will also increase due to retransmissions. The second parameter is the amount of CPU processing needed to run the protocols for data transmission or receipt. For the lossy network example, this amount increases due to the computation involved in handling timeouts and preparing messages for retransmission. One important thing to notice is that the area above B is *not* affected by the amount of time it takes to complete the communication action, i.e. transfer the data; if there is a pause in the data transmission, no energy consumption is attributed to the communication-specific component.

The energy model presented above partitions energy consumption for data transfer into component parts such that each part can be considered independently with respect to its contribution to total energy consumption. Since both machine-specific and device-specific energy consumption only depend on transfer time, it is relatively easy to determine their contribution to total energy consumption. The determination of communication-specific energy consumption is a more difficult problem, since it depends on the configuration of the communication parameters. In Section 7.3, we will discuss one such example for a mobility-oriented transmission protocol.

7.2 Power-Aware Communication

The objective of our work is to reduce the energy usage of communication actions. As stated earlier, mobile hosts operate in dynamic network environments. The transmission-time power management techniques we aim to develop must consider the effects of changes in network services on the performance of underlying transmission protocols, and therefore, changes in energy consumption. Specifically, assuming that there is some fixed level of energy consumed by the simple transfer of data in an error-free environment, as errors are introduced into the data transmission stream, energy consumption increases due to three factors: retransmission and processing of lost messages, additional ACK/NAK transmission and processing, and increased transfer time.

The remainder of this chapter focuses on protocol behaviors in the presence of errors, specifically on the efforts necessary to compensate for lost messages. This is because, in a mobile environment, the possibility of a lost message is much higher than for typical wired environments, and so a protocol that optimizes for lost messages may be more efficient. Our future research may also consider the effects of transmission-time message reordering.

We can attribute transmission-time energy consumption to several protocol behaviors, as discussed in Section 4.4.1. Consider again the breakdown of energy consumption during data transmission, as indicated in Figure 12. Device-specific and machine-specific energy consumption will only be affected by unnecessary wait time, since both are time dependent. Communication-specific energy consumption, on the other hand, is only affected by unnecessary retransmissions and unnecessary acknowledgments. Given this separation of contributing factors, we need to target multiple, and potentially conflicting, parameters.

7.3 Experimental Evaluation

The hypothesis of our research is that power management techniques aimed at a single device may in the end increase the amount of energy consumed by the mobile host for a communication action. The goal of our experiments is to present a specific example of a situation where minimizing communication-specific energy consumption has a detrimental effect on the total energy consumed by the mobile host. These experiments target applications with large data sets to transmit. These sets may be images, sensor data, or simulation data. We target such data to determine the effect

of power management on large data transfers. As a result, our experiments are designed to have the sender actively sending for a long period of time. During this time, we measure the energy consumed by the data transfer. In this section, we use a similar experimental setup to the one presented in Section 4.4.2. We also present the results from our experiments. In Section 7.4, we generalize these results to define techniques for adaptive power management.

7.3.1 Energy Consumption for TCP

The design of TCP and other such transmission protocols has included reliability as well as congestion and flow control. Since such protocols have been developed in the context of the Internet, much effort has been put into congestion detection and recovery as well as compensating for out-of-order message delivery. As a result, TCP has been optimized to react to lost messages as an indication of congestion. In such cases, TCP backs down and reduces its transmissions in an effort to ease the congestion. In contrast, losses in a mobile environment are often caused by interference, not congestion, and so should be aggressively retransmitted without backing down. Additionally, TCP makes an initial assumption that a missing message at the receiver may still be en route and will arrive out of order. Since, in a wireless environment, the probability of message loss is significantly higher than the probability of message reordering, protocols should be optimized for treating missing messages as lost.

Acknowledgments are used by the receiver to indicate to the sender which messages have or have not been successfully received. Acknowledgments can be either positive (ACK), to indicate a successfully received message, or negative (NAK), to indicate a potentially lost message. TCP, for example, uses ACKs to indicate the last successfully received message, and continues to send an ACK for the same message when lost messages have been noticed. In order to compensate for potentially out-of-order messages, TCP delays its ACKs for a period of time. This allows the potentially en route data to arrive without causing an unnecessary retransmission, but delays the retransmission of actual lost messages. Additionally, the sender will not retransmit a message until it has received three duplicate ACKs for the same message, adding additional delay to the retransmission for actual lost messages.

In order to determine the effect of message loss on energy consumption for TCP, we set up a simple experiment run between two mobile hosts using wireless Ethernet devices. We transferred 7MBytes of data using TCP and measured the energy consumed by the whole computer. We simulated lost and corrupted messages by having the receiver randomly drop message at the device driver level. As the number of lost messages increased to 15%, the communication-specific energy consumption increased 20%, but the transfer time (and so the device-specific and machine-specific energy consumption) increased over 500%.

Figure 51 presents the three components of energy consumption for an active TCP connection. The scale on the left hand side is the ratio of the increased energy consumption of the communication action due to loss to the energy consumption of the communication action with no loss. Figure 51(a) shows the communication energy consumption as message loss increases. As we can see, the amount of energy used for data transfer increases at least linearly with the loss rate. This can be expected since the sender needs to retransmit those lost messages. It is important to note here that TCP tends to minimize the number of unnecessary retransmissions. The middle graph in Figure 51(b) shows us the communication-specific plus the device-specific energy consumption. With this graph, we can already see the effect of having to maintain the network interface device in an active mode for an extended period of time: although the energy consumption for transmitting messages has been kept low, the energy consumption for the device has increased significantly.

Now add in the energy used by the rest of the system (Figure 51(b), top graph); note the dramatic change in scale from Figure 51(a) to Figure 51(b). The top and middle graphs appear to increase exponentially with loss rate. The reason is that both the device-specific and machine-specific components are linear in time, while TCP’s congestion-control mechanism, which doubles the retransmission timeout on successive losses, effectively causes the total transfer time to grow exponentially with increasing loss rate.

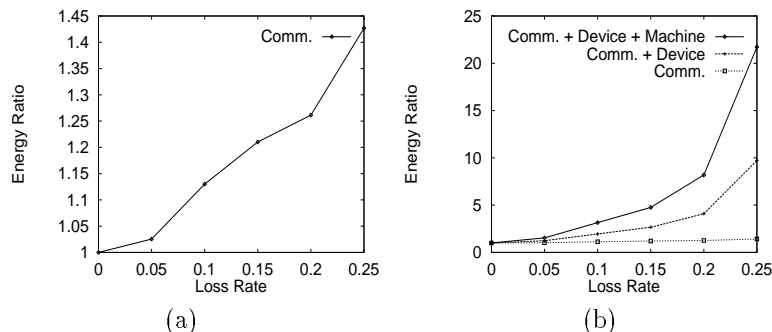


Figure 51: Energy Consumption for TCP

It can be argued that TCP is inefficient for wireless networks where losses of up to 30% can be experienced. Issues with the use of TCP in mobile environments are well-known and are not the subject of our work. The point of this example is simply to illustrate that we need to consider issues beyond identifying the overheads associated with transmitting messages. We next discuss a more wireless-friendly transmission protocol and show similar, though not as dramatic, results.

7.3.2 Energy Consumption for SACK

The previous results with TCP lead us to consider the use of selective acknowledgments (SACKs) to indicate all successfully received messages to the sender [48, 37, 26]. By using a SACK, the receiver can aggressively tell the sender which messages have been received and which messages can be assumed lost or reordered. The use of SACKs also leaves a large amount of flexibility in the design of the sender. The standard sender could be optimized for loss compensation, but, through the monitoring of the packet stream, could be adapted to a more out-of-order friendly protocol in such situations. For the rest of this discussion, we are going to focus on the efforts necessary to compensate for lost messages. In the future, it would be interesting to consider the effect of transmission-time message reordering on such a protocol and its energy consumption.

The goal of these experiments is to demonstrate the tradeoff between minimizing the number of unnecessary retransmissions vs. minimizing the unnecessary wait time and so evaluate the effect of this tradeoff on the total energy consumption of a mobile host. In this section, we consider a mobile host that primarily transmits data.

To demonstrate such an energy tradeoff, we implement a SACK-based transmission protocol. There are two operating modes for this protocol. In the normal case, the protocol sends a simple retransmission when it determines a message has been lost. The second option is to send two retransmissions of the same message back-to-back. This option can significantly reduce transfer time in environments where, if the first retransmission gets lost, the second retransmission still has a good chance of being transmitted successfully. Figure 52 shows the running times of such a protocol for each option. These times are the average of 5 runs of the experiment. Each experiment

transferred a 7MByte block of data, with loss rates varying from 0-25%. As we can see, the use of double retransmissions can reduce the transfer time by 5% of the original transfer time for 10% loss and by 53% for 25% loss. Information about transfer time also allows us to determine both device-specific and machine-specific energy consumption.

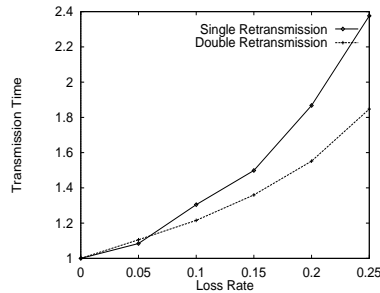


Figure 52: Running Time

To understand how retransmission policy and transfer time translate to energy consumption, we can examine the results similarly to our earlier examination of TCP. Figure 54(a) shows the communication-specific energy consumption for both options of the SACK protocol for a mobile host that is primarily sending data. The results for a receiver are similar and have been omitted for brevity. As we would expect, double retransmissions cost more energy than single retransmissions due to an increase in the number of unnecessary retransmissions. Figure 53 compares the number of unnecessary retransmissions used for each configuration for a 7MByte data transfer. Since transfer time does not play a part in communication-specific energy consumption, the time saved by the use of double retransmissions does not help reduce the communication-specific energy consumption. If we now include the device-specific energy consumption (Figure 54(b)), we can see that the energy consumption for single and double retransmissions has almost evened out. This can be explained by the fact that the device-specific energy consumption is solely dependent on transfer time. Finally, the inclusion of the machine-specific energy consumption shows us that the use of double retransmissions can actually save energy (Figure 54(c)). For both Figure 54(b) and Figure 54(c), the increased energy consumption for single retransmission is due to the increase in unnecessary wait time.

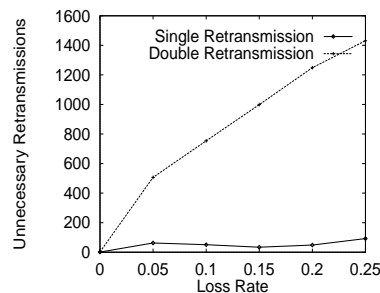


Figure 53: Unnecessary Retransmissions

The experimental results in this section utilize a high-end laptop with high machine-specific run time energy consumption. From these results, we can also predict the energy consumption

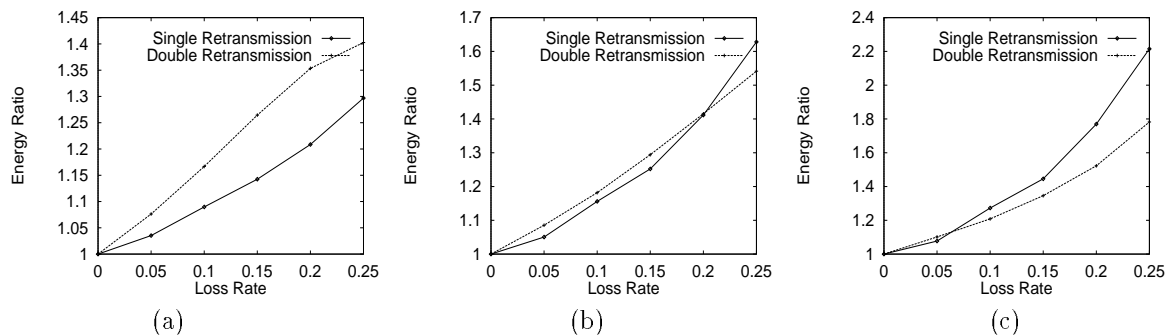


Figure 54: Energy Components

of more energy efficient machines. For example, the previous experiment was performed on the NEC Versa, which consumes around 13.3W when idle. If we consider a machine like the Toshiba Libretto 60 which consumes about 7W when idle, we would expect to see results like those in Figure 55. This demonstrates that our energy model enables power management techniques across machines with different energy consumptions.

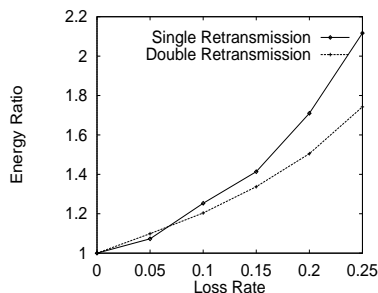


Figure 55: Total Energy Consumption

A limitation of our initial experimental results is the assumption that losses are random. In ongoing research, we use a more realistic loss model, based on the assumption that successive packet losses are correlated. We implemented a packet loss model based on the model presented by Zorzi [73]. In this model, success or failure of an individual packet corresponds directly to the state of a binary Markov channel. For a given packet length and transmission rate, transition probabilities between the two states depend solely on the specified average packet loss and the mobile host's speed. For our experiments, a slow mobile host (i.e., 2-10Km/h) will experience slow fading, which is characterized by long average burst lengths. A faster mobile host (i.e., 25-50Km/h) will experience fast fading, which is characterized by shorter average burst lengths, similar to random loss.

Our initial studies show that for fast fading, the effect of using the SACK protocol with double retransmissions is similar to the results presented above. This is due to the fact that for fast fading the average burst length is quite short. The benefit of using double retransmissions comes from situations where the first retransmission gets lost, but is actually at the end of a loss burst. In this case, and so the second retransmission will be successfully transmitted. In the case of slow fading,

the average loss burst length is quite long, which means that it is unlikely that the first retransmission will be at the end of a loss burst. Double retransmissions for slow fading actually increase the energy consumption, the mobile host is paying the overhead of unsuccessfully transmitting the second retransmission. This is actually the argument used in [72]. This scheme introduces an energy-efficient error control strategy for mobile communications, which delays retransmissions when the channel is impaired. These results show that, since the channel is impaired and it is likely that the retransmission will also get lost, delaying the retransmission improves its chances of success.

These results demonstrate that it is important to have knowledge of the state of the communication channel in order to make intelligent power management decisions. To this end, the next section presents techniques for adaptive power management based on such information.

7.4 Towards Adaptive Power Management

The goal of this research is to design adaptive power management techniques aimed at minimizing the energy consumed during data transfer. The experiments described above demonstrate the effects of different configurations of a transmission protocol with varying channel qualities. Our next step is to use this information to adapt to changes in channel quality. In this section, we consider such power management in the context of the framework described in 2. For this part of the research, we identify relationships among energy consumption and the communication variables captured in this communication framework. Finally, we present a simple method for determining appropriate communication parameters based on the energy consumption associated with specific configurations.

7.4.1 Optimizing Energy Consumption

Through the use of our configurable communication layer, we can configure communication protocols to optimize energy consumption. We can control these parameters to effect such optimizations. When considering transmission protocols, especially error control functionality, these parameters include the techniques used for error control (ARQ vs. FEC) as well as the parameters for these techniques. The parameters for an ARQ-based protocol might be window size or timer values, while they might be redundancy level for FEC. In Section 7.3, we presented a SACK-based transmission protocol with a parameter to control the number of retransmissions to send.

In order to understand how to adjust these parameters, we have developed a communication adaptation algorithm which we have specialized for power management. Communication adaptation to optimize energy consumption involves two stages. First, the correct configuration for a specific point in time is determined, given current network service. Second, network services are monitored for changes in service quality. The effects of such changes on energy consumption are considered to determine a new configuration.

Given a fixed network resource specification, we consider a range of possible operating parameters and choose those that minimize energy consumption. The algorithm in Figure 56 determines minimum energy consumption. We demonstrate our algorithm using FEC as a sample communication protocol, which effects protocol processing, data size and transfer time. We are currently investigating the effect of FEC on energy consumption in the context of our communication framework.

To evaluate individual parameters, we need information about the relationships among the

```

(0) Best =  $\infty$ 
(1) for each RedundancyLevel  $i$  do
(2)   DataSize $_i$  =  $F_{FEC}$ (RedundancyLevel, OrigDataSize)
(3)   CodingTime $_i$  =  $T_{FEC}$ (RedundancyLevel, OrigDataSize)
(4)   TransferTime $_i$  =  $T_{tt}$ (DataSize $_i$ , ChannelState)
(5)   CommunicationEnergy $_i$  =  $E_{FEC}$ (RedundancyLevel, OrigDataSize) +
       $E_{comm}$ (DataSize $_i$ , ChannelState)
(6)   DeviceEnergy $_i$  =  $E_{dev}$ (CodingTime $_i$ , TransferTime $_i$ )
(7)   MachineEnergy $_i$  =  $E_{mach}$ (CodingTime $_i$ , TransferTime $_i$ )
(8)   Energy $_i$  = MachineEnergy $_i$  + DeviceEnergy $_i$  + CommunicationEnergy $_i$ 
(9)   if (Energy $_i$  < Best) then
(10)     level =  $i$ 
(11)     Best = Energy $_i$ 
(12) RedundancyLevel = level

```

Figure 56: Algorithm for Determining Minimum Energy Consumption for Data Transmissions using FEC

communication variables, as discussed in Section 2.1. What we want is a mapping from communication configuration, communication behavior and network service to channel service. In general, such a mapping may be formulated empirically (using profiling techniques prior to running the application or during execution, i.e. based on recent history), or in some cases it may be devised analytically. For example, we could use the experimental results from Section 7.3, which allows us to map retransmission policy (communication configuration) to transfer time (channel service). For the algorithm described in Figure 56, we consider the effect of forward error control on energy consumption. The function F_{FEC} (Step (2)) supplies the mapping from redundancy level and original data size (communication configuration) to transmission data size (communication behavior). T_{FEC} (Step (3)) supplies the mapping from and redundancy level original data size to coding time (channel service) and T_{tt} (Step (4)) supplies the mapping from data size and channel state to transfer time (channel service). The next three steps (Steps (5), (6) and (7)) use the results from Steps (2), (3) and (4) to determine the components of the energy consumption. Finally, the communication framework evaluates the expected total energy consumption associated with communications that operate with these service parameters. These steps are repeated to determine the energy consumption for each possible operating point (as indicated by the loop in Step (1)). From these operating points, the communication framework, which is responsible for power management, chooses the redundancy level that minimizes total energy consumption.

The second stage of communication adaptation reacts to changes in network resources. As such changes are detected, either through information from the network or through online monitoring [56], the communication configuration is reevaluate to determine the effects of these changes on energy consumption.

This section has presented techniques for adaptive power management based on information about the state of the network. A mobile host can use information about the effects of changes in transmission protocol parameters on the actual transmission of the data as well as on the energy consumption of the data transfer. These effects can be determined beforehand, as in the experiments presented in Section 7.3, and used in the optimization algorithm presented above to

minimize energy consumption for data transfer.

7.5 Conclusions

This chapter presents power-aware communication techniques aimed at energy conservation across all components of a mobile host. By understanding the effects of changes in communication parameters on the energy consumption of the entire mobile host, not only the network interface device, we are able to minimize energy consumption through appropriate parameter choices. Previous work was aimed at minimizing the energy consumption of a individual network interface device by regulating retransmissions at the transport level. The results presented in this chapter demonstrate that although the energy consumed during the transmission of the extra messages is not negligible, it is not always significant in the context of the amount of energy consumed by the entire mobile host during data transfer. Our results demonstrate that under lossy network conditions, aggressive retransmission can save a high-end laptop up to 25% over a less aggressive retransmission policy.

With the dramatic increase in the use of mobile computers, the demand for more energy-efficient computers will continue to increase. Due to the diverse uses of these machines, we can only hope to accomplish successful power management through understanding the needs of the individual machines. Inherent in this diversity of use is a diversity of communication requirements, which lends itself to the type of power management techniques described in this chapter.

Chapter 8

Contributions and Future Directions

Applications continue to push the limits of available network bandwidths, processing speeds, available power and other computing resources. Such applications present challenges concerning the specification of service requirements and the online management of service quality for their multiple types of data transfer. Most solutions to guaranteeing services across distributed systems rely on resource reservation [69]. Complementing such research, the objective of our work is to allow applications to operate within some acceptable service specification despite potentially insufficient network resources. Namely, we have developed and evaluated a dynamic interface between the application and the communication layer that permits the communication layer to make application-specific tradeoffs in certain service parameters given its detailed knowledge of current application requirements and communication resources. Such knowledge is kept current by use of dynamic resource monitoring. As a result, application needs and effective communication resource utilization may be optimized jointly.

The dynamic techniques for service management proposed in this thesis address a large class of distributed multimedia applications. These applications share the characteristic that they can tolerate some degree of service degradation, and are often willing to trade off the reduction of certain service requirements against improvements of others. This class of applications includes video conferencing systems, web applications, distributed interactive simulations and distributed virtual environments.

The key innovative contribution of our work is a cooperative solution to dynamic service management: the communication layer uses both application resource requirements and network resource availability to determine how to best configure the application's communications. Specifically, this thesis has presented a cooperative communication layer that configures the operation of its communication protocols based on the perceived benefits to the applications using it. By permitting applications to share dynamic "service quality" requirements with the communication layer, in the form of *payoff functions*, the communication layer is able to make informed decisions about the communication configuration that best suits the current state of the application. Similarly, by dynamically monitoring resource availability from the network and then describing such availability as *service availability curves*, the communication layer is able to make suitable decisions about the communication configuration that best matches the current availability of network resources.

The wide range of communication protocols used by the cooperative communication layer demonstrates the flexibility of our framework and resource management techniques. Specifically, we have presented results from three adaptive protocols. For the variable reliability protocol, we have demonstrated that, by providing detailed information about the quality requirements of the application, the performance of the application can be improved. The cost of such improvements is decreased quality, within an acceptable range, for the specific data. For the power management

protocol, our results demonstrate up to an 83% reduction in the communication-specific and device-specific energy consumption. For a high end laptop, this can translate to a 6-9% savings in the energy consumed by the entire mobile computer. This can represent a savings of up to 40% for current hand-held PCs. The resulting delay introduced is small (0.4-3.1 seconds depending on the power management level). Finally, results attained with our power-aware transmission protocol show that under specific (lossy) wireless network conditions, aggressive retransmission policies can reduce the total energy consumption of the mobile hosts. As losses approach 25%, our techniques show a 25% energy savings for a high-end laptop and predict a 20% savings for a mid-level laptop. With these experimental results, we have validated the concept of applications deriving performance benefits from protocol configuration during the lifetime of the application.

In conclusion, this research has addressed the challenges of dynamic applications operating in dynamic resource environments with results that demonstrate: (1) the need for dynamic configuration of communication protocols, (2) the types of dynamic protocol configurability required by such applications, (3) the opportunities for performance improvements derived from dynamic protocol configuration, and (4) the need for combined information about application requirements and network resource availability to choose acceptable configurations.

8.1 Future Directions

There are several interesting problems associated with this research that are promising avenues for future work. The resource management techniques described for this research depend on many complex components, any of which have potential for new and interesting developments. In this section, we describe a few of the areas that we feel have strong potential for success.

In the context of our power management protocol, open problems include the development of intelligent techniques (e.g., learning-based methods) to estimate when there is queued data at the base station, so that reception delays can be reduced. Such techniques might also adapt the timeout choice for users with varied communication patterns. It would be interesting to explore the correct APIs to provide to applications so that they can give hints to the protocol about their communication patterns (in the spirit of transparent informed prefetching [54]).

The use of variable reliability has shown promise for many different types of multimedia data. We have considered the expansion of our variable reliability protocol to include timing considerations. Although this enhancement was never implemented for this research, the inclusion of timing consideration would result in a useful protocol for many stream-based multimedia applications (e.g., audio or MPEG video). Such a variable reliability protocol enables intelligent retransmission based on knowledge about the structure and timing constraints of the data stream. It would also be interesting to investigate the use of such a protocol to support streaming multimedia applications in lossy network environments.

The results in Chapter 6 give rise to additional and future research concerning suitable payoff or utility functions, suitable times at which payoff functions should be changed by applications using them, and suitable methods for evaluating current or future payoff to applications. Namely, while this research demonstrates the utility of payoff for multimedia applications, we have not yet investigated the practical use of payoff for large-scale and long-running applications in which communication needs change over time. Similarly, while this thesis demonstrates the utility of monitoring and capturing levels of network service, additional research is needed to devise methods for network monitoring, for changing service availability curves over time, and for evaluating the effects of such changes on ongoing communications.

Our current research in the area of power-aware data transmission has been investigating the use

of forward error control (FEC) for wireless data transmission and its effect on energy consumption. In this context, the FEC protocol divides the data into n blocks. It then adds k blocks of redundant data. Of these $n+k$ blocks, the receiver can lose any k blocks and still be able to recreate the original n data blocks. This type of allowable loss fits in perfectly with the design of our variable reliability protocol. It would be interesting to compare the energy consumption of our variable reliability protocol enhanced with FEC to standard transmission protocols for wireless communication.

The resource management techniques in our cooperative communication layer have been designed and implemented for use by a single application with a single data stream. It would be interesting to explore the effects of using our techniques in more realistic and more complex multimedia applications. The determination for how to adapt would be based on global application information that encompasses multiple streams within the application. By considering the value of all data streams used by an application, we would like to investigate the effects of payoff-based adaptation with the goal of allowing the application to continue operating within acceptable limits when resources are limited. The effects of multi-stream quality management can be investigated in the context of a multimedia application that can maintain multiple simultaneous audio, video and data streams originating from multiple distributed hosts. Finally, we believe that our resource management techniques can further be expanded to encompass multiple applications. In this context, the cooperative communication layer would exist on multiple hosts, managing resources across multiple applications.

Appendix A

Configurable Protocol Engine

A.1 Data Structures

```
struct ProtocolBlock :
    ProtocolId pid;
    int len;
    char *hdr;
    int slen;
    char *state;

struct TransportBlock :
    ProtocolId pid;
    int socket;
    int conntype;

struct ConfigBlock :
    ConfigId cid;
    ConfigId recvcid;      ** config id of the otherside
    ConfigId sendcid;     ** config id of this side
    int sender;           ** user id of this side
    int receiver;        ** user id of the other side
    int numProtocols;
    int conntype;
    ProtocolBlock pbs[MAX_PROTOCOL];
    TransportBlock tpb;
    ConfigBlock, *ConfigBlockPtr ;

ConfigBlock *configBlocks;
int numConfig;
myLock configLock;
```

A.2 Functions

A.2.1 Sending Data

```
int sendData (type, *data, len, *label, labellen, sendcid) :
    MsgHdr hdr;
    HdrDesc hd[MAX_PROTOCOL];
    ProtocolBlockPtr pbs[MAX_PROTOCOL];
    long outlen;
    char *sendbuffer;
    char *hdrptr;
    char *descptr;
    char *dataptr = data;
    DataHdr dataHdr;

    allocate sendbuffer
```

```

** start pointing at the beginning of the send buffer
hdrptr = sendbuffer;

initialize header

** we need to send the message in one block, so all hdrs,
** descriptors and data are copied to one buffer.
** this is just a placeholder since we do not have the correct
** header information yet
copy((char *) &hdr, hdrptr, sizeof hdr);
descptr = hdrptr + sizeof hdr;

** create the header descriptors first
for (j = 0; j < number of protocols for cid; j++) do
  set the protocol id and the hdr length

  switch (protId) of
  case JPEG_PROT :
    jpeg_compress(dataptr, &outlen);
    break;
  case VARIABLE_PROT :
    ret = variablesend(state, hdr);
    if (ret == FALSE) then
      abort send and free data buffers
      return;
    end if
    save a pointer to the state and a counter
    break;
  case PMC_PROT :
    if (checkPMCState(state) == FALSE) then
      abort send and free data buffers
      return;
    else
      save a pointer to the state and a counter
    end if
    break;
  case DATA_PROT :
    break;
  end switch

  increment the total length of the descriptors
  copy the descriptor and the hdr
  increment the number of the descriptors
end for

if (labellen != 0) then
  set the protocol id and the hdr length
  increment the total length of the descriptors
  copy the descriptor and the hdr
  increment the number of descriptors
end if

if (sending data) then
  include header for DATA_PROT
  increment the total length of the descriptors
  copy the descriptor and the hdr
  copy the data onto the back of the descriptors
  increment the number of descriptors
end if

if (using Variable reliability protocol) then
  varhold(varState, sendcid, hdrptr, data, varHdr);
else
  if (using PMC protocol) then
    ret = PMCSend(pmcState, pmcHdr, hdrptr, data, outlen, sendcid);
    if (ret == TRUE) then
      send message
    end if
  else
    send message
  end if
end if

```

```

    end if

    free buffers that are not being used
    return 0;
end sendData

```

A.2.2 Receiving Data

```

int recvData (outcid, type, *data, len, *label, labellen, transtype, recvsocket) :

    MsgHdr hdr;
    HdrDesc hd[MAX_PROTOCOL];
    char hdrs[MAX_PROTOCOL][MAX_HDR_LEN];
    HdrDesc newhd[MAX_PROTOCOL];
    char newhdrs[MAX_PROTOCOL][MAX_HDR_LEN];
    char recvbuffer[MAX_BUF_LEN + MAX_DESC_LEN];
    char *hdrptr = recvbuffer;
    char *descptr;

    if (transtype == STREAM_MSG) then
        totalmsglen = readn(recvsocket, &hdr, (sizeof (hdr)));
        *type = hdr.type;

        for (i = 0; i < numDesc; i++) do
            totalmsglen += readn(recvsocket, &(hd[i], sizeof (HdrDesc));

                if (hd[i].len != 0) then
                    totalmsglen += readn(recvsocket, hdrs[i], hd[i].len);

                    if (hd[i].protId == DATA_PROT) then
                        cc = readn(recvsocket, data, hdrs[i]→datalen);
                        len = cc;
                        totalmsglen += cc;
                    else
                        len = 0;
                    end if
                end if
            end for
        else
            totalmsglen = read(recvsocket, recvbuffer, MAX_BUF_LEN + MAX_DESC_LEN);
            bcopy (hdrptr, &hdr, sizeof (hdr));
            descptr = hdrptr + sizeof (hdr);

            for (i = 0; i < numDesc; i++) do
                bcopy(descptr, &hd[i], sizeof (HdrDesc));
                descptr += sizeof (HdrDesc);
                if (hd[i].len != 0) then
                    bcopy (descptr, hdrs[i], hd[i].len);
                    descptr += hd[i].len;
                    if (hd[i].protId == DATA_PROT) then
                        bcopy(descptr, data, hdrs[i]→datalen);
                        len = hdrs[i]→datalen;
                    else
                        len = 0;
                    end if
                end if
            end for
        end if

        if (hdr.recvConfigId == UNINITIALIZED) then
            ** check to see if this sender has sent already.
            hdr.recvConfigId = getcid(hdr.sender, hdr.sendConfigId);
            if (hdr.recvConfigId == UNINITIALIZED) then
                MsgHdr newhdr;
                ** this is a first time send that needs to have the back channel
                ** set up. go through the list of protocols present and
                ** initialize them
                hdr.recvConfigId = createProtocol(hdr.receiver, hdr.sender);
            end if
        end if
    end if

```

```

setRecvCid(hdr.recvConfigId, hdr.sendConfigId);

** send the new cid to the other side for future use
initialize new header
if (transtype == DATAGRAM_MSG) then
    addTransportProtocol(recvConfigId, UDP_PROT, recvsocket);
else
    addTransportProtocol(recvConfigId, TCP_PROT, recvsocket);
end if

for (i = (numDesc-1); i >= 0 ; i-) do
    switch (hd[i].protId) of
        case JPEG_PROT :
            addProtocol(recvConfigId, JPEG_PROT, sizeof (jpeghdr),
                &jpeghdr, sizeof (jpegstate), &jpegstate);
            call individual protocol initialization function
            break;
        case VARIABLE_PROT :
            addProtocol(recvConfigId, VARIABLE_PROT, sizeof (varhdr),
                &varhdr, sizeof (varstate), &varstate);
            call individual protocol initialization function
            break;
        case PMC_PROT :
            addProtocol(recvConfigId, PMC_PROT, sizeof (pmchdr),
                &pmchdr, sizeof (pmcstate), &pmcstate);
            call individual protocol initialization function
            break;
        case LABEL_PROT :
            break;
        case DATA_PROT :
            break;
    end switch
end for
    sendMsg (&newhdr, (sizeof newhdr), newhdr.sendConfigId);
end if
end if

if (*type == NEWCID_MSG) then
    ** this is used to inform the sending side of the receivers cid
    ** the sender cid in the message is the cid from the other side
    setRecvCid(recvConfigId, sendConfigId);
    return 0;
end if

for (i = (numDesc-1); i >= 0 ; i-) do
    switch (hd[i].protId) of then
        case JPEG_PROT :
            if ((len != 0) && (type != BACKCHANNEL_MSG)) then
                getProtocolState(hdr.recvConfigId, hd[i].protId, &jpegstate);
                jpeg_uncompress(data, recvbuffer, len, &jpegstate);
            end if
            break;
        case VARIABLE_PROT :
            if (type != BACKCHANNEL_MSG) then
                getProtocolState(hdr.recvConfigId, hd[i].protId, &varstate);
                ret = variablerecv(varstate, hdrs[i], hdr.recvConfigId, (newhdrs[newhdrcnt]));
                if (ret == TRUE) then
                    newhd[newhdrcnt].protId = VARIABLE_PROT;
                    newhd[newhdrcnt].len = sizeof (VariableHdr);
                    newhdrcnt++;
                end if
            else
                getProtocolState(hdr.recvConfigId, hd[i].protId, &varstate);
                variabeleresp(varstate, hdrs[i]);
            end if
            break;
        case PMC_PROT :
            getProtocolState(hdr.recvConfigId, hd[i].protId, &pmcstate);
            ret = PMCReceive(pmcstate, hdrs[i], &(newhdrs[newhdrcnt]));
            if (ret == TRUE) then

```

```

        newhd[newhdrCnt].protId = PMC_PROT;
        newhd[newhdrCnt].len = sizeof (PMCHdr);
        newhdrCnt++;
    end if
    break;
case LABEL_PROT :
    bcopy(hdrs[i], label, hd[i].len);
    labellen = hd[i].len;
    break;
case DATA_PROT :
    break;
end switch
end for

if (newhdrCnt > 0) then
    MsgHdr newhdr;
    char newsendbuffer[MAX_BUF_LEN + MAX_DESC_LEN];
    char *newhdrptr = newsendbuffer;
    char *newdescptr;

    initialize new header
    bcopy((char *) &newhdr, newhdrptr, sizeof newhdr);
    newdescptr = newhdrptr + sizeof newhdr;

    ** now copy on the new headers.
    for (i = 0; i < newhdrCnt; i++) do
        increment the total length of the descriptors
        copy the descriptor and the hdr
        increment the number of descriptors
    end for
    sendMsg (newhdrptr, (sizeof newhdr + newhdr.desclen), newhdr.sendConfigId);
end if

return len;
end recvData

```


Appendix B

Power Control Protocol

Protocol Commands:

During the course of communication both the master and the slave use commands to inform the receiving side of state changes. Protocol commands are as follows:

- **PMC_CMD_WAKE_UP**: Used by the master to inform the slave that it can wake up and transmit data if it has any messages queued up.
- **PMC_CMD_NO_DATA**: Used by the slave to inform the master that upon wake up, the slave had no data to send.
- **PMC_CMD_DATA**: Used for any messages that simply contains data.
- **PMC_CMD_NEW_DATA**: Used by the slave to indicate to the master that it now has data to transmit.
- **PMC_CMD_DONE**: Used by the slave to indicate the end of data transmission.
- **PMC_CMD_SLEEP**: Used by the master to inform the slave that it should go to sleep.
- **PMC_CMD_SLEEP_OK**: Used by the slave to indicate that it completed the sleep command.

Master Protocol:

The mobile host has the responsibility of determining when communication takes place. At any point in time, the master can be in one of the following states:

- **PMC_STATE_SLEEPING**: The protocol is sleeping and no data can be transmitted or new data requests will be allowed and a **PMC_CMD_WAKE_UP** is sent. Additionally, no data should be received when the protocol is sleeping.
- **PMC_STATE_SENDING**: Only the master is sending data. Subsequent data requests are queued for transmission.
- **PMC_STATE_SENDING_WAIT**: Only the master is sending data. The slave has been queried for new data to send. Subsequent data requests are queued for transmission.
- **PMC_STATE_RECEIVING**: Only the slave is sending data. New data requests will be queued for transmission and the master will enter the **PMC_STATE_SEND_RECV** state.
- **PMC_STATE_SEND_RECV**: Both the master and the slave are sending data. Subsequent data requests are queued for transmission.
- **PMC_STATE_WAITING**: The master woke up and has nothing to send. It has sent a query to the slave to see if it has any new data to send.
- **PMC_STATE_WAIT_FOR_OK**: The master has determined that communication should be suspended and is waiting for a response from the slave.

Slave Protocol:

The base station follows the commands of the master. At any point in time, the slave can be in one of the following states:

- **PMC_STATE_SLEEPING**: The protocol is sleeping and no data can be transmitted. Upon receiving a message, the slave wakes up and enters the **PMC_STATE_RECEIVING** state or the **PMC_STATE_SEND_RECV** state, determined by whether or not the slave has data to send.
- **PMC_STATE_RECEIVING**: Only the master is sending data. New transmissions will not be allowed. If new transmissions are requested, the slave sends a **PMC_CMD_NEW_DATA** message and enters the **PMC_STATE_RECEIVING_WAIT** state.
- **PMC_STATE_RECEIVING_WAIT**: Only the master is sending data. New transmissions will not be allowed. Upon receipt of a **PMC_CMD_WAKE_UP** message, the slave starts transmitting and enters the **PMC_STATE_SEND_RECV** state.
- **PMC_STATE_SEND_RECV**: Both the master and the slave are sending data. Subsequent data requests are queued for transmission.

Appendix C

Power Measurement Details

In order to measure the energy consumed by the laptop, we used the setup shown in Figure 57. In this figure, LT represents the laptop, PS represents the power supply, and T represents the transformer. In order to use measurements for DC currents, we cut the power cord on the laptop side of the transformer. For this particular power cord, there are three lines from the transformer to the laptop. Line 1 is the ground, Line 2 supplies the power to the laptop and Line 3 supplies the power to recharge the battery. We measured V_2 , the voltage drop between Line 2 and the ground line, to be 18.71 volts.

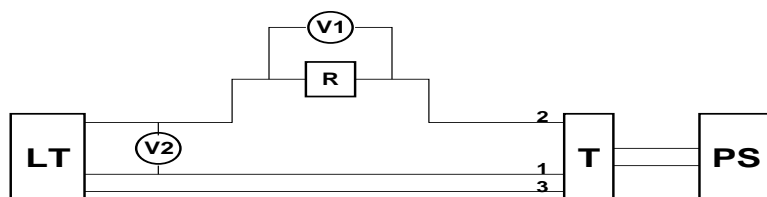


Figure 57: Diagram of Power Measurement Setup

In order to measure the energy consumed during our experiments, we inserted a 0.160 ohm resistor (R in Figure 57) in Line 2 and continuously measured V_1 , the voltage drop across R . We determine the power for the laptop over time from the following equations:

$$I = V_1/R \quad (1)$$

$$P_t = V_2 * I \quad (2)$$

$$Power_t = (P_t * t)/3600 \quad (3)$$

$$Energy = \sum Power_t \quad (4)$$

In Equation 1, we calculate the amps drawn by the computer and in Equation 2, we calculate the power for the instantaneous measurement. Equation 3 converts the instantaneous measurement of P_t to Watt-Hours over that specific measurement period. Finally in Equation 4, we sum over all of the power measurements to calculate the energy consumed during the experiment.

Appendix D

Variable Reliability Protocol

D.1 Sending Protocol

The following are the functions for the sending side of the variable reliability protocol. The application first calls *setupMessage* and then *outgoing message*. These two functions were separated for purely implementation purposes.

```
between (a, b, c):
  if (((a ;= b) and (b ; c)) or
      ((c ; a) and (a ;= b)) or
      ((b ; c) and (c ; a))) then
    return TRUE;
  else
    return FALSE;
  end if
end between

senderInit :
  for each class do
    nextToSend = 0
    ackExpected = 0
    noNak = TRUE
    nBuffered = 0
    nOutstanding = 0
    nextToBuffer = 0
    rclass = 0
    framesBuffered = FALSE
  end for
end senderInit

msgTimeout (msg, len, hdr) :
  ** ok to send message
  for each other class do
    set header value to nextToSend - 1 for class
  end for
  send message
  start message timer
end if
end if
end msgTimeout

checkFrameDone :
  for each class do
    if nextToSend = ackExpected then
      if buffered messages then
        done = FALSE
      end if
    else
      done = FALSE
    end if
  end if
end if
return done
```

```

end checkFrameDone

setupMessage (msg, hdr, newFrame, classState) :
  if newFrame = TRUE then
    ** first to check if all classes are done with the
    ** current frame
    if checkFrameDone(vars) = TRUE then
      ** all classes are done with this frame.
      for each class do
        nextToSend = 0
        ackExpected = 0
        nBuffered = 0
        nOutstanding = 0
      end for
    end if
  end if
end if

if no buffered messages then
  ** set up the header for this message
  hdr:class = class
  hdr:loss = maxconsloss
  for each other class do
    set header value to nextToSend - 1 for class
  end for
end if
end variableSend

outgoingMessage (msg) :
  if nOutstanding  $\geq$  MAX_UNACKED or
  message belongs to next frame then
    ** can't send message
    set id of this message to nextToBuffer
    increment nextToBuffer
    put message on buffer list
  else
    ** try to send message
    ** ok to send message
    set id for this message to nextToSend
    increment nextToSend
    for each other class do
      set header value to nextToSend - 1 for class
    end for
    send message
    start message timer
  end if
end if
end outgoingMessage

processResponse (msg) :
  for each class do
    if response is from this frame
      if NAK message and
      between(ackExpected, hdr:seq + 1, nextToSend) then
        ** process the NAK
        get message from list of sent messages
        ** ok to resend message
        send message
        cancel message timer
        start message timer
      end if
    end if
  end if

  ** process any acknowledgements from this message
  while between(ackExpected, hdr:seq, nextToSend) = TRUE do
    ** acknowledge message with id ackExpected
    cancel message timer
    increment ackExpected
    if buffered messages
      then
        ** the window has opened, so send out a
        ** buffered message if it belongs to this frame
    end if
  end while
end processResponse

```

```

    check next buffered message
    if buffered message belongs to this frame then
        ** ok to send message
        get next buffered message
        decrement nBuffered
        for each other class do
            set header value to nextToSend - 1 for class
        end for
        send message
        start message timer
    else
        decrement nOutstanding
    end if
else
    decrement nOutstanding
end if
end if
end if

** now check to see if we have reached the
** end of a frame so we can move on to the next one
if outstanding frames then
    if checkFrameDone() = TRUE then
        for each class do
            nextToSend = 0
            ackExpected = 0
        end for
        for each class do
            if buffered messages
            then
                ** try to transmit new frame
                while nOutstanding < MAX_UNACKED and
                buffered messages do
                    ** the window is open so try to transmit
                    check next buffered message
                    if message belongs to current frame
                        ** ok to send message
                        get next buffered message
                        decrement nBuffered
                        for each other class
                            set header value to
                                nextToSend - 1 for class
                        end for
                        send message
                        start message timer
                        increment nOutstanding
                    else
                        exit while
                    end if
                end while
            end if
        end for
    end if
end if
end processResponse

```

D.2 Receiving Protocol

The following are the functions for the receiving side of the variable reliability protocol.

```

receiverInit :
    for each class do
        holdPoint = 0
        nextExp = 0
        waiting = 0
        tooFar = MAX_UNACKED
        lossCnt = 0
        conslossCnt = 0
    end for

```

```

    arrived[0:NUM_BUFS] = NOT_RECEIVED
    rclass = 0
end for
end receiverInit

distance (a,b):
    if a < b then
        dist = b - a
    else
        if a > b then
            dist = (VAR_MAX_SEQ_NUM - a + b + 1)
                mod (VAR_MAX_SEQ_NUM + 1)
        else
            dist = 0
        end if
    end if
return dist
end distance

ackTimeout:
    for each class do
        set hdr value to nextToSend - 1 for class
        if NAK then
            set message type to NAK
        else
            set message type to ACK
        end if
    end for

    send message
    start ack timer
end ackTimeout

variablerecv:
    ** process header id for each class looking for
    ** acknowledgements and lost messages
    for each class do
        if new frame then
            holdPoint = 0
            nextExp = 0
            waiting = 0
            tooFar = MAX_UNACKED
            arrived[0:NUM_BUFS] = NOT_RECEIVED
        end if

        ** initialize potential new message to appropriate values
        set hdr value to waiting - 1

        if current class is class of message then
            ** process the incoming data message
            if between(holdPoint, hdr:seq, tooFar) = TRUE)
                and arrived[hdr:seq] != RECEIVED then
                    ** this is a valid message that we have not received yet
                    if between(nextExp, hdr:seq, tooFar) = TRUE then
                        ** this is a new message in the reliability window
                        ** check for losses
                        dist = distance(nextExp, hdr:seq)
                        if dist >= 1 then
                            ** dist indicates the number of lost messages
                            lossCnt = lossCnt + dist
                            ** check to see if the losses put us over either the
                            ** threshold for lost messages in a window or
                            ** consecutive lost messages. Don't check if we
                            ** are already waiting for some message

                            if (waiting.seq = nextExp) and
                                arrived[waiting] != WAITING then
                                    increment conslossCnt
                                    ** find which (if any) message we need to
                                    ** retransmit if we break the maximum
                                    ** loss in window

```

```

    if losscnt > maxloss then
        lossSeq = maxloss + dist - losscnt
    else
        lossSeq = 0
    end if

    ** find which (if any) message we need to
    ** retransmit if we break the maximum
    ** consecutive loss
    if conslosscnt > maxconsloss then
        conslossSeq = maxconsloss + dist - conslosscnt;
    else
        conslossSeq = 0
    end if

    ** if either requirement is broken,
    ** then increment waiting by the
    ** respective amount, differring to
    ** the smaller amount
    if lossSeq < conslossSeq then
        increment waiting by lossSeq
    else
        increment waiting by conslossSeq
    end if
end if
end if

set nextExp to hdr:seq + 1
else
    ** this is a retransmitted message or
    ** a message received out of order
    decrement losscnt
    ** check to see if this is the message we
    ** are waiting for
    if arrived[hdr:seq] = WAITING) then
        conslosscnt = 0
        noNak = TRUE
        ** find the new waiting point
        cnt = 0
        tmpseq = holdPoint
        ** count the current number of losses in
        ** the history window
        while tmpseq != waiting do
            if arrived[tmpseq] != RECEIVED) then
                increment cnt
            end if
            increment tmpseq
        end while
        increment waiting
        ** count the number of losses and losses in a row
        ** in the reliability window
        while waiting != nextExp do
            if arrived[waiting] != RECEIVED then
                increment cnt
                increment conslosscnt
                if cnt > maxloss then
                    exit while
                end if
                if conslosscnt > maxconsloss then
                    exit while
                end if
            else
                conslosscnt = 0
            end if
            increment waiting
        end while
    end if
end if

set arrived[hdr:seq] to RECEIVED

```

```

** check to see if we need to send a NAK message
if losscnt > maxloss or
  conslosscnt > maxconsloss or
  arrived[waiting] = WAITING then
    set arrived[waiting] to WAITING
    if noNak = TRUE then
      ** set up NAK message
      set message type to NAK
      set hdr value for this calcs to waiting - 1
      noNak = FALSE
    end if
  else
    conslosscnt = 0
    waiting = nextExp
    tooFar = waiting + MAX_UNACKED
  end if

** check to see if we can advance the history window
while done = FALSE do
  switch arrived[holdPoint] of
    case RECEIVED:
      catchup = TRUE
      break
    case NOT_RECEIVED:
      ** if the distance between holdPoint and waiting is
      ** larger than the window size, we can advance the
      ** history window
      dist = distance(holdPoint, waiting)
      if dist > MAX_UNACKED then
        done = TRUE
        catchup = FALSE
      else
        ** this message is allowed to be lost
        catchup = TRUE;
      end if
      break
    default
      ** we are waiting on this message
      done = TRUE
      break
  end switch

  if catchup == TRUE then
    ** advance the history window
    if arrived[holdPoint] != RECEIVED then
      ** this was an allowed loss
      decrement losscnt
    end if
    set arrived[holdPoint] to NOT_RECEIVED
    increment holdPoint
  end if

  if holdPoint = waiting then
    done = TRUE
  end if
end while
else
  ** invalid message
end if
else
  ** check to see if there were lost messages in the other classes.
  if arrived[hdr:seq] = NOT_RECEIVED then
    ** we did not receive this message, check if it is valid
    if between(nextExp, hdr:seq, tooFar) = TRUE then
      ** check to see if this loss broke any requirements
      dist = distance(nextExp, hdr:seq)
      losscnt = losscnt + dist + 1;
      nextExp = hdr:seq + 1

      if arrived[waiting] != WAITING then
        conslosscnt = conslosscnt + dist + 1

```

```

** find which (if any) message we need to retransmit
** if we break the maximum loss in window
if losscnt > maxloss then
    lossSeq = maxloss + dist + 1 - losscnt
else
    lossSeq = 0
end if
if conslosscnt > maxconsloss then
    conslossSeq = maxconsloss + dist + 1 - conslosscnt
else
    conslossSeq = 0
end if
** if either requirement is broken, then increment
** waiting by the respective amount, differring
** to the smaller amount
if lossadd < conslossadd then
    increment waiting by lossSeq
else
    increment waiting by conslossSeq
end if
end if

** check to see if we need to send a nak
if losscnt > maxloss or
    conslosscnt > maxconsloss or
    arrived[waiting] = WAITING then
    if noNak = TRUE then
        set message type to NAK
        set arrived[waiting] to WAITING
        set header value to waiting - 1
        noNak = FALSE
    else
        conslosscnt = 0
        waiting = nextExp
        tooFar = waiting + MAX_UNACKED
    end if
end if
end if
end if
end for

if new response message then
    send response message
end if
    start acknowledgement timer
end variablerecv

```


Bibliography

- [1] Tarek Abdelzaher and Kang Shin. End-host architecture for QoS-adaptive communication. In *IEEE Real-Time Technology and Application Symposium (RTAS'98)*, June 1998.
- [2] Elan Amir, Steven McCanne, and Randy Katz. Receiver-driven bandwidth adaptation for light-weight session. In *ACM Multimedia '97*, November 1997.
- [3] Elan Amir, Steven McCanne, and Hui Zhang. An application level video gateway. In *ACM Multimedia '95*. 255-265, 1995.
- [4] Ajay Bakre and B.R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *IEEE International Conference on Distributed Computing Systems (ICDCS) '95*, 1995.
- [5] Hari Balakrishnan, Venkata Padmanabhan, Srinivasan Seshan, and Randy Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *Proceedings of the SIGCOMM '96 Symposium*, 1996.
- [6] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy Katz. Improving TCP/IP performance over wireless networks. In *First ACM International Conference on Mobile Computing and Networking (MOBICOM)*, November 1995.
- [7] B.B. Bederson, J.D. Hollan, A. Druin, D. Rogers, J. Stewart, and D. Vick. A zooming web browser. In *Multimedia Computing and Networking (MMCN) '96*, 1996.
- [8] R. Bettati and A. Gupta. Dynamic resource migration for multiparty real-time communication. In *IEEE International Conference on Distributed Computing Systems (ICDCS) '96*, May 1996.
- [9] Nina Bhatti and Richard Schlichting. A system for constructing configurable high-level protocols. In *Proceedings of the SIGCOMM '95 Symposium*, August 1995.
- [10] T. Bihari and K. Schwan. A comparison of four adaptation algorithms for increasing the reliability of real-time software. Technical report, Department of Computer and Information Science, The Ohio State University, OSU-CISRC-4/88-TR13, April 1988.
- [11] Thomas Bihari and Karsten Schwan. Dynamic adaptation of real-time software. *ACM Transactions on Computer Systems*, 9(2):143–174, May 1991.
- [12] J-C. Bolot. Cost-quality tradeoffs in the internet. *Computer Networks and ISDN Systems*.
- [13] N. Brownlee, C. Mills, and G. Ruth. Traffic flow measurement: Architecture. Technical Report RFC 2063, Internet Engineering Task Force, January 1997.
- [14] Kenneth L. Calvert. Beyond layering: Modularity considerations for protocol architectures. In *Proceedings of the International Conference on Network Protocols (ICNP) '93*, September 1993.

- [15] K.L. Calvert, R.H. Kravets, and R.D. Krupczak. An extensible end-to-end protocol and framework. Technical Report GIT-CC-96-15, Georgia Institute of Technology, 1996.
- [16] Jyh-Chang Chen, Krishna Sivalingam, Prathima Agrawal, and Shaline Kishore. A comparison of MAC protocols for wireless local networks based on battery power consumption. In *Proceedings of IEEE INFOCOM 98*, 1998.
- [17] Imrich Chlamtac, Chiara Petrioli, and Jason Redi. Energy conservation in access protocols for mobile computing and communication. *Microprocessors and Microsystems Journal*, 1998.
- [18] David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the SIGCOMM '90 Symposium*, pages 200–208, September 1990.
- [19] K. Curewitz, P. Krishnan, and J. S. Vitter. Practical prefetching via data compression. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 257–266, May 1993.
- [20] Luca Delgrossi, Christian Halstrick, Ralf G. Herrtwich, Frank Hoffmann, Jochen Sandvoss, and Barbara Twachtmann. Reliability issues in multimedia transport. In *First IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS) '93*, 1993.
- [21] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair-queueing algorithm. *Journal of Internetworking Research and Experience*, pages 3–26, October 1990.
- [22] Jayanta Dey, James Kurose, and Don Towsley. On-line processor scheduling for a class of iris real-time tasks. *ACM Transactions on Computer Systems*, 45(7), July 1996.
- [23] F. Dougliis. On the role of compression in distributed systems. *ACM Operating Systems Review*, 27(2):88–93, April 1993.
- [24] F. Dougliis, P. Krishnan, and B. Bershad. Adaptive disk spindown policies for mobile computers. In *Proceedings of the Second USENIX Symposium on Mobile and Location Independent Computing*, April 1995.
- [25] F. Dougliis, P. Krishnan, and B. Marsh. Thwarting the power hungry disk. In *Proceedings of the 1994 Winter USENIX Conference*, January 1994.
- [26] Kevin Fall and Sally Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *Computer Communications Review*, July 1996.
- [27] Fengmin Gong and Guru Parulkar. An application-oriented error control scheme for high speed networks. Technical Report TR 92-37, Washington University, St. Louis, 1992.
- [28] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *First ACM International Conference on Mobile Computing and Networking (MOBICOM)*, 1995.
- [29] Zygmunt Haas. A protocol structure for high-speed communication over broadband ISDN. *IEEE Network Magazine*, pages 64–70, January 1991.

- [30] D. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Second ACM International Conference on Mobile Computing and Networking (MOBICOM)*, 1996.
- [31] J. Huang and P.-J. Wan. On supporting mission-critical multimedia applications. In *Proceedings of the International Conference on Multimedia Computing and Systems '96*, 1996.
- [32] Norman C. Hutchinson and Larry L. Peterson. The x -Kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.
- [33] Intel Corporation, Microsoft Corporation and Toshiba Corporation. *Advanced Configuration and Power Interface Specification*, revision 1.0a edition, July 1998.
- [34] E. Douglas Jensen, C. Douglass Locke, and Hideyuki Tokuda. A time-driven scheduling model for real-time operating systems. In *IEEE Real-Time Systems Symposium*, December 1985.
- [35] Ad Kamerman and Leo Monteban. WaveLAN-II: A high performance wireless LAN for the unlicensed band. *Bell Labs Technical Journal*, Summer 1997.
- [36] S. Keshav, C. Lund, S. J. Phillips, N. Reingold, and H. Saran. An empirical evaluation of virtual circuit holding time policies in ip-over-atm networks. In *Proceedings of IEEE INFOCOM 95*, 1995.
- [37] S. Keshav and S.P. Morgan. SMART retransmission: Performance with overload and random losses. In *Proceedings of IEEE INFOCOM 97*, 1997.
- [38] Robin Kravets, Ken Calvert, P. Krishnan, and Karsten Schwan. Adaptive variation of reliability. In *the Seventh IFIP Conference on High Performance Networking (HPN'97)*, April 1997.
- [39] Robin Kravets, Ken Calvert, and Karsten Schwan. Dynamically configurable communication protocols and distributed applications: Motivation and experience. Technical Report GIT-CC-96-16, Georgia Institute of Technology, 1996.
- [40] Robin Kravets, Ken Calvert, and Karsten Schwan. Payoff adaptation of communication for distributed interactive applications. *Journal on High Speed Networking: Special Issue on Multimedia Communications*, 1998.
- [41] Robin Kravets and P. Krishnan. Power management techniques for mobile communication. In *Fourth ACM International Conference on Mobile Computing and Networking (MOBICOM'98)*, 1998.
- [42] P. Krishnan, P. Long, and J. S. Vitter. Adaptive disk spindown via optimal rent-to-buy in probabilistic environments. In *Proceedings of the Twelfth International Machine Learning Conference*, July 1995.
- [43] P. Krishnan, P. M. Long, and J. S. Vitter. Learning to make rent-to-buy decisions in probabilistic environments. In Armand Prieditis and Stuart Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*. Morgan Kaufmann, 1995.
- [44] K. Li, R. Kumpf, P. Horton, and T. Anderson. A quantitative analysis of disk drive power management in portable computers. In *Proceedings of the 1994 Winter USENIX*, 1994.

- [45] J. R. Lorch and A. J. Smith. Reducing processor power consumption by improving processor time management in a single-user operating system. In *Second ACM International Conference on Mobile Computing and Networking (MOBICOM)*, 1996.
- [46] Lucent Technologies. *WaveLAN/PCMCIA Card User's Guide*, October 1996.
- [47] Rahmi Marasli, Paul Amer, and Phillip Conrad. Retransmission-based partially reliable services: An analytical model. In *Proceedings of IEEE INFOCOM 96*, 1996.
- [48] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgement options. Request for Comments RFC 2018, Internet Engineering Task Force, October 1996.
- [49] B. Narendran, J. Sienicki, S. Yajnik, and P. Agrawal. Evaluation of an adaptive power and error control algorithm for wireless systems. In *IEEE International Conference on Communications (ICC'97)*, 1997.
- [50] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and R. Walker Kevin. Agile application-aware adaptation for mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles (SOSP) '97*, October 1997.
- [51] Seiwoong Oh, Hiroyuki Sugano, Kazutoshi Fujikawa, Toshio Matsuura, Shinji Shimojo, Masatoshi Arikawa, and Hideo Miyahara. A dynamic QoS adaptation mechanism for networked virtual reality. In *Proceedings of Fifth IFIP International Workshop on Quality of Service*, May 1997.
- [52] S. W. O'Malley and L. L. Peterson. A dynamic network architecture. *ACM Transactions on Computer Systems*, 10:110–143, May 1992.
- [53] Christos Papadopoulos and Gurudatta Parulkar. Retransmission-based error control for continuous media applications. In *The 6th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV) '96*, 1996.
- [54] R. H. Patterson, G. A. Gibson, and M. Satyanarayanan. A status report on research in transparent informed prefetching. *ACM Operating Systems Review*, (27):21–34, April 1993.
- [55] Thomas Plogemann, Bernhard Plattner, Martin Vogt, and Thomas Walter. Modules as building blocks for protocol configuration. In *Proceedings of the International Conference on Network Protocols (ICNP) '93*. Swiss Federal Institute of Technology Zurich, September 1993.
- [56] Daniela Rosu, Karsten Schwan, Sudhakar Yalamanchili, and Rakesh Jha. On adaptive resource allocation for complex real-time applications. In *18th IEEE Real-Time Systems Symposium, San Francisco, CA*. IEEE, December 1997.
- [57] John M. Rulnick and Nicholas Bambos. Mobile power management for maximum battery life in wireless communication networks. In *Proceedings of IEEE INFOCOM 96*, 1996.
- [58] J. Saltz, G. Edjlali, et al. Data Parallel Programming in an Adaptive Environment. *Proc. of the 9th International Parallel Processing Symposium*, pages 812–819, 1995.
- [59] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

- [60] Ashwin Sampath, P. Sarath Kumar, and Jack Holtzman. Power control and resource management for a multimedia CDMA wireless system. In *The Sixth International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'95)*, 1995.
- [61] Douglas C. Schmidt, Donald F. Box, and Tatsuya Suda. Adaptive: A dynamically assembled protocol transformation, integration, and evaluation environment. *Concurrency: Practice and Experience*, June 1993.
- [62] Karsten Schwan and Hongyi Zhou. Dynamic scheduling of hard real-time tasks and real-time threads. *IEEE Transactions on Software Engineering*, 18(8):736–748, August 1992.
- [63] Scott Shenker. Fundamental design issues for the future internet. *IEEE Journal on Selected Areas in Communications*, 13(7), September 1995.
- [64] Krishna Sivalingam, Jyh-Chang Chen, Prathima Agrawal, and Shaline Kishore. Design and analysis of low-power access protocols for wireless and mobile ATM networks. *ACM/Baltzer Journal on Special Topics in Mobile Networking and Applications (MONET)*, June 1998.
- [65] Mark Stemm and Randy Katz. Reducing power consumption of network interfaces in hand-held devices. In *Third International Workshop on Mobile Multimedia Communications (MoMuc-3)*, December 1996.
- [66] Charles J. Turner and Larry L. Peterson. Image transfer: An end-to-end design. In *Proceedings of the SIGCOMM '92 Symposium*, August 1992.
- [67] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of the First Symposium on Operating System Design and Implementation (OSDI '94)*, November 1994.
- [68] Carey L. Williamson and David R. Cheriton. Loss-load curves: Support for rate-based congestion control in high-speed datagram networks. In *Proceedings of the SIGCOMM '91 Symposium*, 1991.
- [69] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, pages 8–18, September 1993.
- [70] John A. Zinky, David E. Bakken, and Richard D. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 1997.
- [71] Martina Zitterbart, Burkhard Stiller, and Ahmed N. Tantawy. A model for flexible high-performance communication subsystems. *IEEE Journal on Selected Areas in Communications*, 11(4), May 1993.
- [72] Michele Zorzi and Ramesh Rao. Error control and energy consumption in communications for nomadic computing. *IEEE Transactions on Computers (Special Issue on Mobile Computing)*, March 1997.
- [73] Michele Zorzi, Ramesh Rao, and Laurence Milstein. Error statistics in data transmission over fading channels. *IEEE Transactions on Communications*, 46:1468–1477, November 1998.

Vita

Robin Kravets received her BS in Computer and Information Sciences from the University of Massachusetts, Amherst in 1989. After working in software development in the Boston area for two years, she moved to Los Angeles. In LA, she worked at a small start up company and attended the University of California in Los Angeles for an MS degree in Computer Science. After graduating in 1993, she started in the Ph.D. Program at the College of Computing, Georgia Institute of Technology. During her time at Georgia Tech, she received an AT&T/Lucent Technologies Ph.D. Fellowship and worked for two summers at Bell Labs.