CROSS-LAYER ENERGY-EFFICIENT MOBILE SYSTEM DESIGN

BY

ALBERT FRED HARRIS III

B.A., Rockhurst University, 2000
B.S., Rockhurst University, 2000
M.S., University of Illinois at Urbana-Champaign, 2002

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

To my wonderful family Erin, Kane, and the new one that is on the way... and all our future family members.

Abstract

Advances in computing technology have resulted in high-end mobile devices with extensive computational capabilities that enable complex applications such as video encoding or lossless data compression. However, the energy requirements of such applications far exceed current battery technology, which is not improving as quickly as computing capabilities. Techniques have been designed targeting each component of the mobile system. However, conserving energy in any single system component in isolation may ultimately result in a net increase in the energy consumption of the whole system because it does not take into account resource usage in different components.

To deal with the mismatch between the rapidly increasing energy demands of mobile multimedia systems and the slow increase in battery capacity, research is being focused on energy efficient mobile system design. Through the use of a system energy model that captures the tradeoffs between adaptations in various system components, we design a cross-layer adaptive system, including a MAC protocol, a transport protocol, and a suite of applications. The adaptive MAC layer algorithm, Pincher, achieves energy efficient communication through fast data transmission. Using the mechanisms incorporated in Pincher, we analyze the impact of such adaptations on total system energy consumption. This analysis shows that information about bandwidth must be used by higher layer adaptation algorithms to provide energy efficient adaptations.

The adaptive reliability transport protocol, Reaper, supports adaptive applications with various timing and reliability requirements. This transport protocol exposes information about energy costs, data rate available, and expected packet loss rate for use in application adaptations.

Finally, the components of the system architecture are tested using adaptive applications. We have developed an adaptive file transfer application and an adaptive video encoding application. This suite of applications is used to show that greater energy savings can be achieved by using a cross-layer adaptive system than by using discrete adaptations in single layers of the system.

Acknowledgments

My family has been amazing in supporting me and putting up with my late nights working. My friends have been wonderful, helping to take care of my family and giving me the support I need. I could not have done it without all of them. Finally, this thesis would not have been possible without the help of help of all my group members, Cigdem Sengul, Robin Snader, and Yaling Yang, our advisor Robin Kravets, and friends who graduated before me, Luiz Magalhaes and Seung Yi.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Advances in computing technology have resulted in high-end mobile devices with extensive computational capabilities that enable complex applications such as video encoding or lossless data compression. However, the energy requirements of such applications far exceed current battery technology, which is not improving as quickly as computing capabilities. This mismatch has resulted in the design and development of energy conservation mechanisms in all components of a mobile system, including CPU [24], network [22], OS [54], disk [19], display [27] and applications [9, 33]. However, conserving energy in any single system component in isolation does not take into account the effects on resource usage in different components and may ultimately result in a net increase in the energy consumption of the whole system.

Energy, like any other resource, must be managed effectively to ensure that its consumption results in applications providing the services required by the user. In general, resource management problems have been solved at the system level by using a layered model. Essentially, each layer is in charge of a particular set of resources. Management decisions about these resources can then be made locally within each layer, without the need for information transfer across layers. This shields each layer from needing detailed information about adaptations that may be taking place in other layers. However, this layered approach does not directly fit managing energy in a mobile system. The challenge stems from the fact that energy cannot be managed in isolation from other resources since the use of other resources in the system generally consumes energy.

While energy conservation lends itself well to a monolithic solution, where a global resource manager knows the details of each mechanism in the system, such monolithic solutions are inefficient. The inefficiencies come from the need for complete knowledge of the details of all mechanisms

throughout the system to make intelligent global allocation decisions. However, complete knowledge of each adaptation mechanism throughout a system would require collecting very large amounts of data about all mechanisms at frequent intervals. To avoid such overhead, energy conservation needs to be integrated into a layered system. The challenge lies in finding a way to manage the information transfer between layers. Due to the cross-layer interactions of adaptation mechanisms on energy consumption, finding what information to share and how often to share that information is not straightforward. Furthermore, blindly passing information between layers is ineffective and inefficient. Passing too much information wastes time and energy. Passing too little information or the wrong information makes it impossible to make correct adaptation decisions consistently. Passing stale information also leads to incorrect adaptation decisions. Therefore, to support incorporating energy constraints into adaptation mechanisms, it is first necessary to identify what information should be passed between layers through modeling and profiling of adaptation mechanisms. Essentially, any change in an adaptation mechanism that causes significant fluctuations in energy consumption in the system needs to be exposed to other layers. To understand and expose the causes of such fluctuations, it is necessary to model and profile the individual layers and mechanisms. Such modeling can ultimately show the interactions between layers and reveal important thresholds that can be used for cross-layer adaptation triggers.

While conserving energy can increase the lifetime of a device by draining its battery store at a slower rate, the ultimate goal must be to conserve energy while still providing meaningful services to the user. Although the network could trivially conserve energy by turning off the interface card and never transmitting any data, this would not support the goal of providing services to the user. For bulk data, a meaningful service can be defined as transferring all of the data reliably. For multimedia communications, defining what is meaningful is not as simple. Traditionally, a concept of *utility* has been used as a metric measuring the usefulness of data. Once utility is defined, the goal for system optimizations is to minimize energy consumption while maintaining some threshold level of utility. While defining utility functions is outside the scope of this work, to evaluate our system performance, we use a common definition of utility for multimedia data (see Chapter 6).

The final component in achieving energy efficient systems is to design adaptation algorithms to support effective communication for the applications while conserving energy. The difficulty

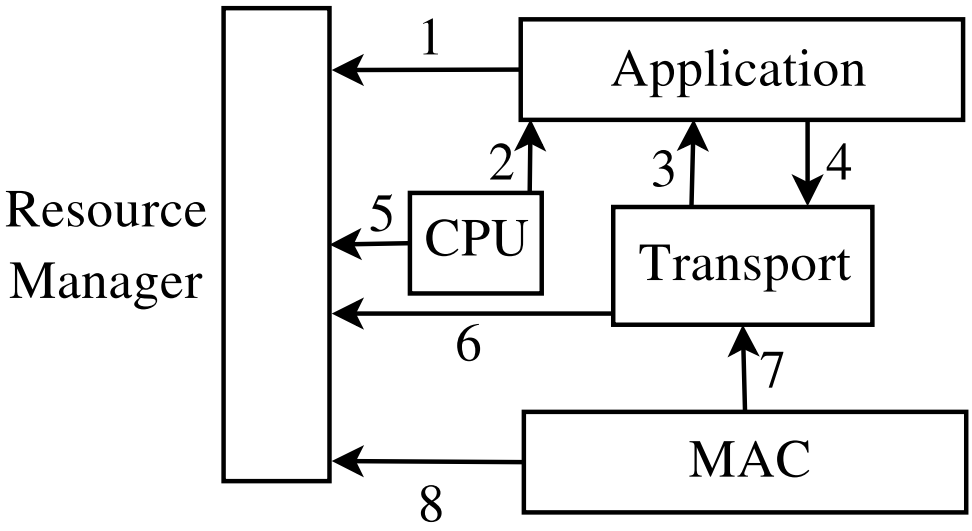

Figure 1.1: System Interactions

is that mechanisms traditionally used in a single layer, which perform optimally in the absence of energy constraints, often perform sub-optimally in the face of energy constraints. Therefore, new mechanisms are required for each layer of the system that are capable of making use of the cross-layer information that is now available.

This thesis presents our design, implementation, and evaluation of a comprehensive, energy-aware wireless system. The system is designed in the context of wireless nodes connected to a basestation, communicating with wired nodes in the Internet. Therefore, it focuses only the energy spent by the wireless node and assumes that the wired nodes are not energy constrained.

The major contributions of this work are, first, a rigorous exploration of the adaptation space in which such mobile systems exist. This exploration defines what information must be passed between layers to facilitate effective and efficient cross-layer adaption techniques. It, furthermore, reveals when such information passing is required. Second, we designed a framework for passing information between the layers. Third, our adaptive algorithms perform cross-layer adaptations, demonstrating how such information should be used to conserve energy. Finally, we present a verification of our research by using real applications on top of our adaptive mechanisms.

1.1 Adaptive Mobile Systems Design

A number of approaches to adaptive mobile systems design have been used in recent research. This thesis is part of the GRACE project [3], which aims to design a cross-layer hierarchical adaptive mobile system. Others have used similar approaches. Fugue [8], developed CPU and operating system level adaptation [24, 54], or designed middleware to handle application adaptations based on resource levels [9, 33]. This is different than the work in this thesis insofar as it does not take the further step of how to integrate the general cases of adaptations in all layers in a coordinated fashion across multiple applications.

Figure 1.1 gives an overview of the system presented in this thesis with each communication pathway numbered. The particular choices of pathways and information to flow along the pathways is motivated and explained in detail throughout the rest of the thesis. The application layer (Section 6) shares information with both the resource manager and the transport layer of the network (paths 1 & 4). The application shares information about resource requirements in terms of CPU and network usage for use in resource management decisions. The application also shares information about playout buffers, frames rates, frames sizes, and loss tolerances with the network transport layer. The CPU shares information with both the resource manager and the application about resource availability with the resource manager and about resource cost (in terms of energy per cycle) to the application. The transport layer (Section 4) shares information about estimated throughput with the resource manager and information about estimated throughput, loss rate, frame loss information, and network cost (in terms of energy per bit) with the application. The MAC layer (Section 3) shares information about current bit rates with the resource manager and information about current network costs (in terms of energy per bit) and MAC layer frame drops with the transport layer.

1.2 Thesis Overview

We start the presentation of our complete system with a comprehensive system energy model in Chapter 2. This model allows the mapping of the adaptation space for adaptation mechanisms in each system layer, exposing interactions between adaptive mechanisms throughout the layers of the

system.

Chapter 3 presents the design of an adaptive network layer. Our network layer adaptation algorithm, called Pincher, uses two adaptation mechanisms to provide energy efficient communication. Through the use of these mechanisms, Pincher maintains the best possible stream quality while maintaining energy efficient communication. We demonstrate that faster data transmission results in the most energy efficient communication for wireless LAN technologies, allowing Pincher to operate without information from other system layers. However, because Pincher reacts to signal quality in the environment, changing the available bandwidth for the system, information about Pincher's adaptations must be exposed to other layers of the system. Pincher, therefore, exports information about the current data rate and cost in terms of energy per byte for use in adaptations in other layers.

Chapter 4 analyzes the space of adaptations available to transport protocols with specific focus on multimedia stream reliability. First, we analyze various adaptation mechanisms and their effects on energy consumption. Then, using insights from this analysis, we designed Reaper, an adaptive, energy-aware transport protocol that supports cross-layer adaptation. Reaper supports various reliability levels and the use of timing constraints to support both bulk data applications and multimedia applications to allow for different types of tradeoffs (*e.g.*, a video application could tradeoff reliability requirements for coding cost). Applications are then able to choose an appropriate level of reliability for a given stream and adapt that reliability while the stream is active. Reaper achieves significant gains in terms of the number of good application frames received and the amount of energy consumed for the transmission of that data.

Chapter 5 presents our resource manager, Corvus, which supports cross-layer resource information sharing. The Corvus resource manager allows adaptation mechanisms in each layer of the system to expose information about adaptations and resource costs, allowing other mechanisms to use this information.

Chapter 6 presents the evaluation of our cross-layer adaptive mechanisms, showing that significant energy savings can be obtained by exposing and using information from all layers of the system in adaptation algorithms. To show this result, we use a suite of applications including an adaptive file transfer application and an adaptive video encoding application. The adaptive file

transfer application (aFTP) dynamically adapts to changes in resource availability and cost based on knowledge of the relative contribution of energy consumption from each layer of the system but requires 100% reliability from the transport layer. The adaptive video application (aVE) also adapts the amount of compression used on a frame based on the relative contribution of energy consumption for each layer of the system. The video stream can also operate with some loss in the data stream. We use these applications to demonstrate that intelligent use of energy consumption information from multiple layers of the system can be leveraged to save energy beyond what can be saved at each layer alone. Chapter 7 ends with some conclusions and future directions for this research.

Chapter 2

Energy Consumption In Mobile Devices

Energy consumption in mobile devices stems from all layers of the system (*e.g.*, CPU, network, display), each of which have been targeted for energy conservation. The specifics of where energy is consumed in each layer can only be examined by an in depth energy model. Such a model allows the complex cross-layer relationships between different components of the system to be captured. In general, the specific triggers that cause adaptations in a system must be identified. In this thesis, we show that two main resource categories drive adaptations: energy (*e.g.*, the amount of battery power left) and available system resources (*e.g.*, the amount of network bandwidth available, CPU).

The model presented in this chapter explicates which changes in the system cause large energy fluctuations and so allows decisions about what data needs to be shared across layers. Also, such modeling highlights potential dependencies across layers (*i.e.*, mechanisms that may effect other layers' energy consumption). Therefore, this chapter presents a system-wide energy model that allows intelligent decisions to be made about what data to expose between components across layers of the system. This model includes network and CPU energy, relating the two together as application energy consumption. Essentially, the energy model drives the creation of effective interfaces between system components, which we present in Section 5.1.

2.1 Network Energy Consumption

Network energy can be divided into three distinct parts: energy to transmit, energy to receive, and idle energy. Idle-time energy consumption depends on the power usage of the interface while

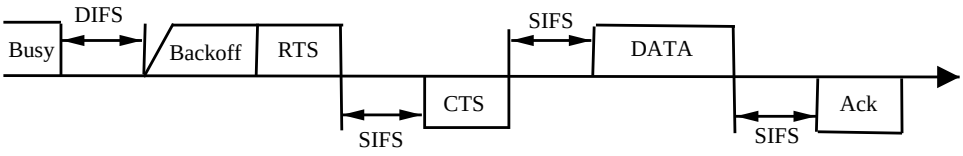

Figure 2.1: The IEEE 802.11 RTS/CTS Protocol

idle and the amount of idle time. Idle-time energy consumption can be reduced by placing the interface into a low-power sleep state. However, while idle-time power management is outside the scope of this thesis, our work complements idle-time power management by maximizing the amount of time the interface could be put to sleep. We specifically model communication time network energy consumption, of which there are two components: transmit time and receive time. For IEEE 802.11 wireless LANs, transmit and receive time energy consumption can be considered at a per-data-frame level. Therefore, to completely model energy, both control mechanisms and data must be considered. IEEE 802.11 uses an RTS/CTS protocol for channel negotiation, depicted in Figure 2.1. Each part of the RTS/CTS protocol consumes energy for each packet sent.

2.1.1 Transmission Energy Consumption

From the perspective of a sender, once the medium has become quiet, the sender must stay in receive mode for DIFS time, then for a backoff, then an RTS is sent. The sender must be in receive mode for SIFS time and during the reception of the CTS packet. Then the sender stays in receive mode for SIFS time and sends the data frame. Finally, the sender must remain in receive mode for SIFS time plus the time to receive the acknowledgment. Therefore, the total sender energy per frame, assuming no loss is, given by:

$$\begin{aligned} E_{networkSend} \equiv &(E_{DIFS} + E_{backoff} + E_{rts-send}) + (E_{SIFS} + E_{cts-recv}) + \\ &(E_{SIFS} + E_{data-send}) + (E_{SIFS} + E_{ack-recv}). \end{aligned} \tag{2.1}$$

The energy consumption during the DIFS interval (E_{DIFS}) is given by the DIFS time interval, which is about 52μs, and the power consumption of the wireless interface card while in receive mode.

$$E_{DIFS} \equiv t_{DIFS} \times P_{recv} \tag{2.2}$$

The energy consumption during the SIFS interval (E_{SIFS}) is given by the SIFS time interval, which is about 44μs, and the power consumption of the wireless interface card while in receive mode.

$$E_{SIFS} \equiv t_{SIFS} \times P_{recv} \tag{2.3}$$

The energy consumption during the backoff period ($E_{backoff}$) is given by the backoff time and the power consumption of the wireless interface card while in receive mode. The specific amount of backoff energy is difficult to estimate but easy to measure. Backoff time is directly proportional to the amount of contention for the medium. Therefore, contention for the wireless channel will increase communication energy costs by increasing the average backoff period.

$$E_{backoff} \equiv t_{backoff} \times P_{recv} \tag{2.4}$$

The energy to send the RTS ($E_{rts-send}$) is determined by the size of the RTS (D_{rts}), the data rate at which the RTS is sent (R), the transmit power level (P_t), the cost to keep the interface card in transmit mode (P_{xmit}), and the base cost of the interface card (P_{base}).

$$E_{rts-send} \equiv \frac{D_{rts}}{R} \times (P_t + P_{xmit} + P_{base}) \tag{2.5}$$

The energy consumption to receive the CTS is given by size of the CTS ($D_{cts-recv}$), the data rate at which the CTS is sent (R), and the power consumption of the interface card while in receive mode (P_{recv}).

$$E_{cts-recv} \equiv \frac{D_{cts}}{R} \times P_{recv} \tag{2.6}$$

Potentially the largest component of energy consumption is the transmission of the actual data. This is the main focus of the rest of the thesis. The energy consumption to send the data ($E_{data-send}$) is determined by the size of the data frame (D), the size of the header information (D_{header}), the data rate at which the frame is sent (R), the transmit power level (P_t), the cost to keep the interface card in transmit mode (P_{xmit}), and the base cost of the interface card (P_{base}).

$$E_{data-send} \equiv \frac{(D + D_{header})}{R} \times (P_t + P_{xmit} + P_{base}) \tag{2.7}$$

P_{base} and P_{xmit} are fixed by the interface device. In the context of Equation 2.7, energy conservation techniques aim to affect one or more of the other components. For example, for a given set of channel characteristics, TPC minimizes P_t, while BPM affects R.

The energy consumption to receive the acknowledgment ($E_{ack-recv}$) is determined by the size of the acknowledgment (D_{ack}), the data rate at which the acknowledgment is sent (R), and the power consumption of the interface card while in receive mode (P_{recv}).

$$E_{ack-recv} \equiv \frac{D_{ack}}{R} \times P_{recv}. \tag{2.8}$$

Since the wireless medium is inherently lossy, a complete energy model must include the energy consumption form loss recovery. There are four problems during the RTS/CTS protocol in which a loss can cause the sender to retransmit data. First the RTS or CTS packets could be lost. From the perspective of the sender, these events appear the same. Second, the data or ACK packets can be lost. Again, these two events appear the same to the sender. It is important to model the energy cost to recover from these losses to understand the cost of MAC layer retransmission in both cases.

The loss of an RTS packet essentially means the sender loses the work done during the backoff and the RTS send, plus the energy expended waiting for the CTS to timeout ($t_{cts-timeout} \times P_{recv}$).

$$E_{cost-rts-loss} \equiv (t_{cts-timeout} \times P_{recv}) + E_{backoff} + E_{rts-send} \tag{2.9}$$

The loss of a data frame essentially means the sender loses the work done from the backoff period to the data being sent, plus the energy consumption waiting for the ACK to timeout ($t_{ack-timeout} \times P_{recv}$).

$$\begin{aligned} E_{cost-data-loss} \equiv & ([t_{ack-timeout} \times P_{recv}] + E_{backoff} + E_{rts-send}) + (E_{cifs} + E_{cts-recv}) + \\ & (E_{cifs} + E_{data-send}) \end{aligned} \tag{2.10}$$

The complete energy model allows a understanding of what network adaptations impact the energy consumption of the network in such a way as to need to be exposed to the application. Also, the application can only affect the amount of data to be sent and therefore, $E_{data-send}$.

2.1.2 Receive Energy Consumption

One final component of active communication is receive energy consumption. Since this thesis is mostly concerned with sender optimizations, the receive energy consumption model is presented for completeness. From the perspective of the receiver, the first event in the RTS/CTS protocol is the receipt of an RTS. Therefore, the total energy to receive is determined by the energy to receive the RTS ($E_{rts-recv}$), the energy consumption during the SIFS interval (E_{SIFS}), the energy consumption to send the CTS ($E_{cts-send}$), the energy to receive the data frame ($E_{data-recv}$), and the energy to send the acknowledgment ($E_{ack-send}$).

$$E_{networkRecv} \equiv E_{rts-recv} + (E_{SIFS} + E_{cts-send}) + (E_{SIFS} + E_{data-recv}) + (E_{SIFS} + E_{ack-send}) \quad (2.11)$$

The energy consumption to receive the RTS is determined by the size of the RTS (D_{rts}), the data rate at which the RTS is sent (R), and the power consumption of the interface card while in receive mode (P_{recv}).

$$E_{rts-recv} \equiv \frac{D_{rts}}{R} \times P_{recv} \quad (2.12)$$

The energy consumption to send the CTS ($E_{cts-send}$) is determined by the size of the CTS (D_{cts}), the data rate at which the CTS is sent (R), the transmit power level (P_t), the cost to keep the interface card in transmit mode (P_{xmit}), and the base cost of the interface card (P_{base}).

$$E_{cts-send} \equiv \frac{D_{cts}}{R} \times (P_t + P_{xmit} + P_{base}) \quad (2.13)$$

The energy consumption to receive the data frame ($E_{data-recv}$) is determined by the data frame size (D) plus the header size (D_{header}), the data rate at which the frame is sent (R), the power consumption of the interface card while in receive mode (P_{recv}).

$$E_{data-recv} \equiv \frac{(D + D_{header})}{R} \times P_{recv} \quad (2.14)$$

The energy consumption to send the acknowledgment ($E_{ack-send}$) is determined by the data size of the acknowledgment (D_{ack}), the data rate at which the acknowledgment is sent (R), the transmit power level (P_t), the cost to keep the interface card in transmit mode (P_{xmit}), and the

base cost of the interface card (P_{base}).

$$E_{ack-send} \equiv \frac{D_{ack}}{R} \times (P_t + P_{xmit} + P_{base}) \tag{2.15}$$

2.2 CPU Energy Consumption

CPU energy consumption (E_{CPU}) can be modeled in one of two ways, depending on whether or not dynamic voltage scaling (DVS) is used. We do not use DVS in this work; therefore, we use a constant cost per cycle for the CPU energy consumption and can model the CPU costs, where P_{CPU} is the CPU energy consumption in mJ/cycle, as follows:

$$E_{CPU} = I_c \times P_{CPU}. \tag{2.16}$$

The inclusion of DVS results in a model of the form: $E_{CPU} \propto C_{eff} \times V^2 \times I_c$, where V is the voltage, f is the frequency, C_{eff} is the effective capacitance, and I_c is the number of cycles used by the application. Our framework supports the use of such models, which are more complicated than Equation 2.16, but still relate the energy consumption of the CPU to the number of cycles used during execution.

2.3 Application Energy Consumption

Application energy consumption is defined by how much the application uses the CPU and how much data it passes to the network. Therefore, the application energy consumption, E_{app}, is defined in terms of the amount of energy consumed by the network to send the data, $E_{data-send}$, the amount of energy consumed by the CPU to run any algorithms used by the application, E_{CPU}, and the costs associated with disk, I/O, memory, display, etc., E_{other}.

$$E_{app} = E_{data-send} + E_{CPU} + E_{other} \tag{2.17}$$

For applications using techniques such as adaptive compression or video encoding, there is an interesting relationship between $E_{data-send}$ and E_{CPU}. Optimizations, such as compression,

aimed at decreasing the amount of data transmitted, and so $E_{data-send}$, typically require more processing and so increase E_{CPU}. Therefore, application-driven energy conservation requires cross-layer information to react to dynamic conditions in the network and CPU. For the purposes of this thesis, we assume E_{other} is constant. However, as future work outside the scope of this thesis, it would be interesting to incorporate the effects of adaptive techniques that affect E_{other}, such as disk spin-down algorithms.

2.4 Contributions and Future Directions

This section presents a unified, comprehensive system energy model explicating system wide energy relationships among energy-efficient mechanisms used in the remainder of this work.

While the energy model for the purposes of this thesis is complete, future directions involve incorporating specific CPU energy models. Current technology in adaptive CPU and memory work allow sections of the processor and memory system to be shut down when not needed. Such architecture adaptations are not captured by the DVS CPU model presented in Section 2.2. However, the model as a whole can easily incorporate more complex models for the CPU.

Chapter 3

MAC Layer Protocol

Wireless communication protocols must have the ability to maintain communication in the face of poor signal quality. MAC layer protocols are the first group to look at. They control the rate (R) and the transmit power (P_t) of the interface card. Two main techniques have been studied to combat interference in wireless communications. Transmit power control (TPC) [11, 16] increases the transmitted signal strength to essentially "speak louder" than the interference. Bandpass modulation adaptation (BPM) [15, 22, 28, 41] adds redundancy to the transmitted signal to allow the receiver to recover from errors in the signal due to interference. However, neither of these techniques are free in terms of energy.

The key challenge to achieving minimum energy consumption while maintaining communication that can tolerate the interference in the channel stems from the inter-relationship between transmission rate and transmission cost. This relationship requires solutions that jointly optimize the choice of bandpass modulation scheme and transmit power level. In general, transmission-time energy consumption is determined by the time for transmission, as defined by the rate at which the data is sent multiplied by the power to drive the transmission. For wireless communication, transmission rate is determined by the bandpass modulation scheme used. To support communication in diverse environments, current wireless interfaces support multiple bandpass modulation schemes, which are differentiated by the amount of noise they can tolerate. Essentially, higher noise tolerance results in lower data rates. While the choice of available bandpass modulation schemes is based purely on the characteristics of the wireless channel, the resulting data rates define the time needed for transmission.

The power needed to drive the transmission can be attributed to three components of a wireless

interface: P_t used to determine the signal strength of the transmission, the base cost to activate the card (P_{base}), and the cost to drive the transmitter (P_{xmit}). Many cards support dynamic adaptation of transmit power level through TPC, which is used to attempt to reduce the P_t and so reduce the total energy required for transmission. Such changes in P_t affect the signal strength at the receiver and therefore affect the possible choice of bandpass modulation schemes and rates. This, in turn, impacts the transmission-time energy consumption. The final two components, P_{base} and P_{xmit}, are fixed by the specific interface card used and so their contribution to total energy consumption is only affected by the time needed for transmission. For example, choosing a bandpass modulation scheme with a slower data rate (R) increases the amount of time for transmission. Although this may allow P_t to be lowered, the amount of energy consumed by driving the transmitter (as opposed to leaving it idle) can outweigh the savings of using a lower P_t, increasing the overall energy consumption for the transmission.

Transmission-time energy consumption is determined by the amount of time that the device is transmitting and the specific settings of the device's transmitter. Many wireless network cards provide multiple rates at which data can be transmitted. The adaptation of the transmission rate (R) is called *bandpass modulation adaptation* (BPM). Essentially, the chosen R determines the time needed to send the data. The specific R used is determined by a number of factors, including the *signal-to-noise ratio* (SNR) and the target reliability of the transmission [17]. For the receiver to correctly receive the packet, the SNR must be greater than a certain threshold (SNR_t). As long as the receiver's SNR is maintained above this threshold, P_t at the sender can be reduced, directly reducing energy consumption at the sender.

The goal of joint BPM and TPC is to minimize energy consumption by looking at all possible values for each and choosing the most efficient combination. The bandpass modulation scheme defines a minimum SNR that must be maintained, where SNR represents the tolerance to interference. Each available bandpass modulation scheme can be coupled with a P_t that achieves the necessary SNR for a given interference level. There are three ways to pick a bandpass modulation scheme, the slowest [29, 38], the fastest, or something in between [40].

Typical wireless network devices support multiple modes (*i.e.*, transmit, receive, idle, sleep). While the transmit and receive modes are driven by the transmission of data, a significant amount

of energy can be conserved by transitioning the device into a sleep mode instead of remaining idle [52, 45, 4, 37]. The benefits come from the magnitude of the difference between the energy costs when idle (*e.g.*, approx 1W) and when sleeping (e.g., approx 95 mW) [7]. In this thesis, we do not make use of a power management scheme that can take advantage of low power sleep modes. However, our energy model accounts for such techniques and our approach supports easy integration of idle-time power management. The design and integration of power management techniques is part of future work.

3.1 Wireless Communication

The goal of any wireless networked communication is to provide a service to the application that tolerates the interference in the channel and minimizes the cost to the system. The cost of the communication, in terms of energy, is used to define the amount of effort put into providing communication. The SNR at the receiver, as defined by $\frac{S}{N}$, effectively limits the possible bandpass modulation schemes that can function given the interference in the channel. Transmission signal strength, S in dB, is determined by P_t used by the sender and the distance between the sender and the receiver. Noise, N in dB, is a measure of ambient signals originating from other radio sources transmitting in the same band. Each bandpass modulation scheme has a threshold SNR (SNR_t), below which it cannot support communication.

Although the level of noise is not controllable by either party, the sender can increase S, and so increase the SNR, by increasing P_t for the transmission. Additionally, noise tolerance can be added to the transmission by encoding data bits using multiple raw bits. However neither of these techniques is free, in terms of bandwidth, latency, or energy. The relationship between these channel characteristics and cost can be captured by the energy efficiency of the communication. Energy efficiency (ξ) in bits per joule is defined as the ratio of the number of application bits received at the receiver (D) in bits to the energy used in the transmission of the data ($E_{data-send}$) in joules:

$$\xi \equiv \frac{D}{E_{data-send}}, \tag{3.1}$$

where D is defined by the amount of data to send. However, $E_{data-send}$, and so ξ, is governed by

the techniques used to compensate for interference in the channel. Although multiple techniques can be used for interference compensation in wireless networks, we focus on two main techniques: TPC and BPM. In the remainder of this section, we analyze the complex relationship between ξ and these error compensation techniques.

Compensation for interference is typically performed at the physical layer. However, the physical layer is complex and can be further divided into three distinct sub-layers, two of which support compensating for poor quality wireless links. *The Transmit/Spreading sub-layer* controls the transmitter, through parameters such as channel frequency (*e.g.*, 2.4GHz, 5GHz) and P_t. For a given wireless interface, the channel frequency is predefined. If the wireless interface has TPC capabilities, P_t can be dynamically adjusted. *The Multiplexing sub-layer* is responsible for sharing the medium. Possible multiplexing schemes include TDMA and CSMA. For a given wireless interface, the multiplexing scheme is predefined. *The Bandpass Modulation sub-layer* is responsible for encoding the transmission stream into the actual signal being transmitted. The data throughput of the wireless channel is determined by the coding scheme used. Some common bandpass modulation schemes are DBPSK and DQPSK. If the wireless interface supports BPM, the specific coding scheme can be dynamically chosen.

Although sub-layer 2 does not support adaptation, both sub-layers 1 and 3 can adapt components that impact the effectiveness and energy efficiency of the communication channel. BPM dynamically chooses from a set of coding schemes based on the current error characteristics of the channel. While a more aggressive bandpass modulation scheme can compensate for more bit errors, it also increases the amount of time needed to transmit the data and so can increase the amount of energy consumed. TPC adapts P_t used for communication to match S to the required SNR_t. While a reduced P_t can reduce the energy consumption for transmission, it may also result in a reduced R. Although each of these techniques can be used in isolation, the intelligent combination of the two can result in more energy efficient communication. To evaluate the effect of combining these two techniques, we first present each individually and then discuss how to combine them.

3.1.1 Bandpass Modulation Adaptation

BPM compensates for noise on a wireless channel by encoding application layer bits into "raw" bits to be sent through the air. The number of raw bits per application data bit, or the coding rate, along with other code-specific overhead, defines the transmission rate (R) of the channel. More efficient codes introduce less redundancy and so achieve higher data rates. However, higher rate codes generally tolerate less noise than lower rate codes. This tolerance can be captured by SNR_t defined for each specific scheme.

Since the choice of bandpass modulation scheme determines R, it also impacts energy efficiency. For a given bandpass modulation scheme, the energy to transmit the data is dictated by R, D, and P_t (see Equation 2.7). For a fixed P_t, it is clear that higher values of R yield a lower $E_{data-send}$ since D is not affected by the choice of bandpass modulation scheme. Since ξ is inversely proportional to $E_{data-send}$ (see Equation 3.1), and so directly proportionally to R, higher values of R yield more energy efficient communication.

To compensate for variations in interference levels in wireless links, wireless LAN communication supports a range of bandpass modulation schemes. For example, IEEE 802.11b supports four schemes: 1Mbps (DBPSK), 2Mbps (DQPSK), 5.5Mbps (CCK), and 11Mbps (CCK). Current protocols for selecting a bandpass modulation scheme focus on choosing the fastest rate given the currently perceived channel quality, typically measured in terms of SNR. Ramanathan and Steenstrup [41] present a dual-channel slotted-aloha medium access control (MAC) protocol that uses a separate control channel over which the receiver transmits feedback about losses encountered to the sender. The sender then chooses the bandpass modulation scheme to use based on the loss rate. Auto Rate Fallback (ARF) [28] adapts IEEE 802.11 to allow the sender to pick the best rate based on channel quality information from the last data packet. Gass, *et. al.* [15] propose a protocol for point-to-point links that selects rates based on cached, per link information about the previous channel quality. Vaidya, *et. al.* [22] propose a receiver-side rate adaptation scheme in which the receiver picks the best rate at which to send during the RTS/CTS packet exchange in the IEEE 802.11 protocol. They show that using receiver-side information improves the performance of the rate-adaptation algorithm. The general assumption of all of this work is that the sender has no control over the SNR at the receiver and so chooses the most effective R, which is essentially the

fastest R that can tolerate the current SNR.

3.1.2 Transmission Power Control

TPC compensates for noisy channels by altering P_t of the wireless interface with the goal of changing the SNR at the receiver to match the SNR_t of the chosen bandpass modulation scheme. As P_t increases, S at the receiver increases, improving the quality of the channel. However, higher values of P_t increase the energy cost of communication. The energy efficiency (ξ) of a TPC scheme is directly related to the P_t chosen for the data transmission. In the case of TPC alone, R is constant and lower values of P_t result in lower values of $E_{data-send}$ (see Equation 2.7). Since ξ is inversely proportional to $E_{data-send}$, lower values of P_t result in increased ξ. Therefore, given a fixed R, the lowest P_t that results in a sufficient SNR for a given bandpass modulation scheme yields the highest energy efficiency.

Given the shared characteristics of wireless communication channels, any node within transmission range of the receiver can interfere with reception. The use of heterogeneous transmit power levels among the nodes can lead to an increase in collisions. However, using homogeneous transmit power levels for the RTS/CTS and ACK, these collisions lead to minimal performance degradation in wireless LAN environments [40].

3.1.3 Combination BPM & TPC

To combine BPM and TPC, the interactions between the two techniques must be understood. The goal of TPC is to choose the lowest possible P_t that will not violate some SNR_t. However, such thresholds are dependent on a particular bandpass modulation scheme. Therefore, a bandpass modulation scheme must be chosen before TPC can be used to set P_t to match the SNR_t. When choosing a bandpass modulation scheme, there are three possibilities: always choose the slowest scheme, choose varying schemes based on some algorithm, or always choose the fastest scheme. To understand the impact of these options for choosing the bandpass modulation scheme on ξ, it is necessary to look into the relationship between the capacity of the wireless channel and P_t.

Shannon's law can be used to define the relationship between channel capacity and S at the receiver (see Equation 3.2), where C is channel capacity in bits per second, B is bandwidth in

Hertz, S is in decibels and N is noise level in decibels [50]:

$$C = B \times log_2(1 + \frac{S}{N}). \tag{3.2}$$

However, C represents the raw capacity of the channel. To capture the achievable data rate, R, it is necessary to include the characteristics of the coding scheme. If α represents the overhead of the coding scheme (with values between zero and one, zero representing the highest overhead), the relationship between C and R can be stated as:

$$R = \alpha \times C. \tag{3.3}$$

This relationship captures the fact that coding schemes with higher values of α achieve lower values of R given the raw capacity of the channel.

The first possibility for choosing a bandpass modulation scheme is to pick the slowest one possible, as hinted at by Prabhakar, *et. al.* [38], who use these relationships to choose the most energy efficient R and P_t. They note that if B and N are held constant, S is monotonically decreasing and convex in $\frac{1}{C}$. This implies that lower channel capacities require lower signal strengths. If a fixed α is assumed, C defines R, implying that lower values of R require lower values of S. Finally, since S is directly related to the P_t used by the sender, lower values of R require lower values of P_t [38].

While this implies that the slowest data rate is the most energy efficient, these results are theoretical and cannot be directly applied to the coding schemes used in wireless LAN environments. First, algorithms for choosing a bandpass modulation scheme and P_t must react to changes in the noise in the environment. Consider a single bandpass modulation scheme used in IEEE 802.11b networks, DQPSK, that achieves a data rate of 2 Mbps. For DQPSK to operate correctly, a certain SNR_t must be maintained, which implies a minimum channel capacity (see Equation 3.2). Therefore, as N increases, P_t must be increased to increase S and raise C to an acceptable level. This curve is monotonically decreasing and convex in $\frac{1}{N}$, showing that as N increases, the energy per bit to successfully transmit increases. This shows the direct tradeoff between transmission-time energy consumption and using TPC to maintain channel state.

Second, each bandpass modulation scheme has its own α value. A key assumption for Prabhakar,

et. al., is that values of R are defined by P_t and so reducing P_t directly reduces R. However, for current wireless LAN environments, values of R are changed by changing bandpass modulation schemes, each of which has a different α. Additionally, actual interface cards have a discrete number of transmit power levels available. This, coupled with the fact that higher rate bandpass modulation schemes use more efficient encodings, and so have lower values of α, leads to the fact that always using the slowest rate bandpass modulation scheme does not result in the most energy efficient communication. Consider a scenario where N in the environment is very low, 10^{-20} decibels, and the minimum P_t is 1mW. The minimum P_t defines the minimum possible channel capacity given the noise level according to Equation 3.2. For Prabhakar, *et. al.*, this in turn defines R. However, actual wireless LAN environments use different bandpass modulation schemes to achieve different values of R. Therefore, while an optimal channel capacity may have been located by using the lowest P_t, the *transmission rate* has not. Consider an IEEE 802.11b card with 11Mbps CCK, 5.5Mbps CCK, 2Mbps DQPSK, and 1Mbps DBPSK bandpass modulation schemes. With a noise level so low, the channel capacity is large enough to handle any of these bandpass modulation schemes. However, the transmitter must be on for a longer time at exactly the same P_t for the lower values of R. Therefore, slower rates do not result in energy efficient communication.

It is clear then that when the capacity of the channel is sufficient for any of the bandpass modulation schemes to operate at the lowest P_t, the fastest R is cheapest. Therefore, slower rates are not always the most energy efficient. There are noise levels that require P_t to be raised above the minimum for the fastest bandpass modulation scheme to be used. If the α values in Equation 3.3 were the same for each bandpass modulation scheme, then it would turn out that the slowest rate at the lowest power level would be cheaper. However, the modulation schemes used in wireless LAN devices do not have such constant α values. Again, consider the environment in the last scenario, now with available transmit power levels of 5mW, 20mW, 30mW, 50mW, and 100mW. If N in the environment is 1.8dB, the channel capacity can sustain 1Mbps and 2Mbps rates at 5mW. At 20mW it can sustain 5.5Mbps and 11Mbps. Because the energy consumption is four times greater at the higher rates, but the rates themselves are 5 to 10 times greater, the fastest rate results in the most energy efficient communication.

Kompella, *et. al.* [29] attempt to apply the conclusions from Prabhakar *et. al.* to a real network

card. They choose the slowest rate bandpass modulation scheme that prevents any data playout times from being violated. However, our analysis above shows that this direct application to actual wireless interfaces does not work.

The second possibility for choosing a bandpass modulation scheme is to use different rate schemes depending on the characteristics of the communication. Miser [40] attempts to choose the most energy efficient R/P_t combination for each packet by calculating a table offline containing R/P_t combinations indexed on factors such as how many times a particular packet has been retransmitted, etc. This results in varying choices of bandpass modulation schemes, not necessarily the fastest or the slowest. However, their approach in not practical since the network configuration must be known at the time the table containing the R/P_t combination is calculated offline. This network configuration information must indicate the number of contending stations and determine the RTS collision probability.

The final possibility for choosing a bandpass modulation scheme is to always choose the fastest available. This approach cannot be supported if only the energy consumption due to P_t is considered. However, it is important to note that none of the above solutions consider that slower packet transmission requires the wireless interface to be in transmit mode for a longer amount of time. Once these costs are considered, we show that fast data transmission yields the most energy efficient communication.

3.2 System Communication Costs

Most current approaches to energy conservation in wireless networking focus on the energy costs as dictated by the energy consumed by the transmitter. However, the transmitter is not the only component of the wireless interface that consumes energy and is typically not even the most significant component. Therefore, it is necessary to understand the specific components that contribute to energy consumption and how BPM and TPC affect these components.

Recall the components of transmission energy for a wireless LAN card from Equation 2.7, P_{base}, P_{xmit}, and P_t. With CSMA-based protocols, the wireless interfaces are required to continuously sense the medium. Therefore, P_{base} is equal to the idle costs. Energy conservation techniques that use TPC only affect P_t, however, P_t accounts for only one part of P_{total}. Since P_t is typically on

the order of 10mW to 100mW and P_{base} and P_{xmit} are typically on the order of 1W [7], the cost of transmission is dominated by the energy consumed by the other components of the transmission costs. Therefore, solutions only targeting P_t minimize the smallest component of the total transmission costs.

A wireless communication device consumes energy when it is idle or listening to the channel as well as when it is actively communicating. Such idle costs can dominate the energy consumption of a node, especially if there is not much active communication. Power management techniques are aimed at reducing idle-time P_{base} by turning off some or all of the components of the interface card. To use these modes effectively, power management algorithms place the wireless interface in sleep mode based on the communication patterns of the wireless interface [13, 31, 45, 52]. The key component of power management relevant to a power-saving communication protocol is that the longer the device is placed in a low-power mode, the more energy is saved, indicating that faster data transmission increases the energy efficiency of communication.

3.3 Fast Data Transmission

The benefits from fast data transmission stem from the relationship between P_{base}, P_{xmit}, and P_t. However, the energy consumption of any wireless interface depends on the specific values of each. Since current wireless interfaces have a wide range of values, we present our analysis of fast data transmission on a common IEEE 802.11b wireless interface. We then explore the impact of potential improvements in wireless interfaces to determine the range of cards for which fast data transmission is the most energy efficient. Our analysis shows that major breakthroughs in wireless communication are necessary before something other than fast data transmission needs to be considered.

For our initial evaluation, we use a Cisco Aironet 350 series PC Card [7], with available bandpass modulation schemes of 1Mbps, 2Mbps, 5.5Mbps, and 11Mbps and available transmit power levels of 5mW, 20mW, 30mW, 50mW, and 100mW. The energy consumption of the Cisco Aironet in the different card modes are: 2.24W for transmit, 1.35W for recv/idle, and 0.075W for sleep. For this card, the transmission energy costs (P_{total}) are broken down as follows: $P_t = [5 - 100]$mW,

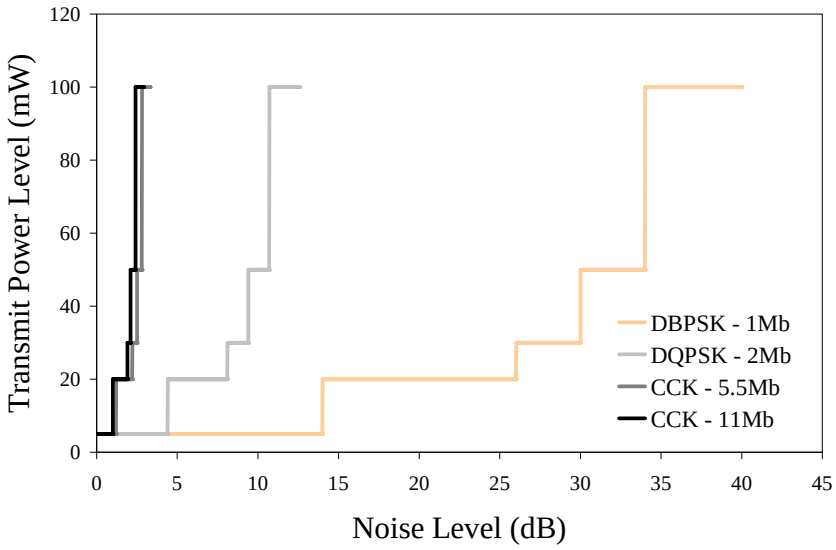

Figure 3.1: Transmit Power Levels per Noise level

$P_{xmit} = 890$mW and $P_{base} = 1275$mW. Therefore, recall Equation 2.7

$$E_{data-send} = (\frac{D}{R}) \times P_t + P_{xmit} + P_{base}.$$

To completely evaluate the effect of fast data transmission on energy efficiency, it is necessary to look at each individual component of energy consumption during communication. Therefore, we first look at the characteristics of the Cisco Aironet card based on the available bandpass modulation schemes and transmit power levels. Next, we evaluate the energy consumption from the energy consumed by the transmit power level. We then incrementally reintroduce the cost of driving the transmitter and the base cost of the card. From our evaluation, we determine how much better wireless interfaces need to get before it is necessary to consider schemes other than fast data transmission.

3.3.1 Wireless Interface Capabilities

Wireless interfaces have discrete bandpass modulation schemes and transmit power levels. Each bandpass modulation scheme has a certain relative N that can be tolerated for each P_t. These thresholds for the Cisco Aironet card are depicted in Figure 3.1. The x-axis represents the noise at the receiver (N) in dB and the y-axis represents P_t in mW. The values of P_t for each bandpass mod-

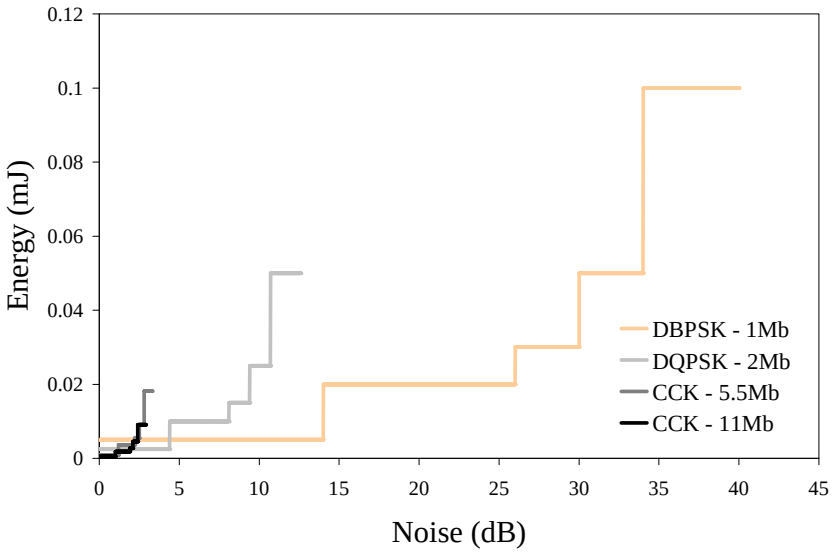

Figure 3.2: Energy Consumption Considering Only Transmit Power Level

ulation scheme are represented by step functions because there are discrete bandpass modulation schemes and transmit power levels supported by the interface card. For each bandpass modulation scheme, there is a relative noise level that can no longer be tolerated, these points occur where each line ends (*e.g.*, at noise 2.9dB the CCK at 11Mbps terminates). This maps out the space in which a BPM/TPC algorithm must operate for the Aironet card. For a more detailed evaluation of the space in which BPM/TPC operates across different technologies, see Appendix C.

3.3.2 Transmission Energy

To evaluate the impact of fast data transmission on the energy consumption defined by P_t, we evaluate the transmission energy ($E_{data-send}$) with P_{xmit} and P_{base} set to zero (see Equation 2.7) for each bandpass modulation scheme for D = 1Kb. $E_{data-send}$ for each bandpass modulation scheme is represented by the step functions in Figure 3.2. The x-axis represents N in dB and the y-axis represents $E_{data-send}$ in mJ. Clearly, ignoring all other costs, the most energy efficient communication uses the bandpass modulation with the lowest $E_{data-send}$. Because the faster bandpass modulation schemes cannot tolerate values of N greater than 3dB, Figure 3.3 represents a close up of Figure 3.2. Recall that there are three basic strategies for choosing a bandpass modulation scheme: always choose the slowest scheme, choose varying schemes, or always choose the fastest scheme. When considering only P_t, always choosing the slowest bandpass modulation scheme is

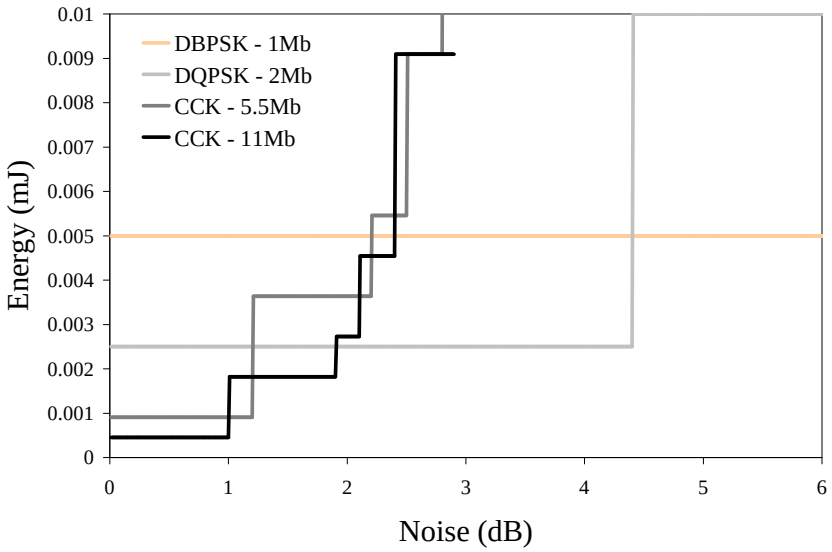

Figure 3.3: Close-up: Energy Consumption Considering Only Transmit Power Level

not the most efficient, as can be seen in Figure 3.3. The line representing DBPSK, the slowest bandpass modulation scheme, results in consistently higher values of $E_{data-send}$ until the fastest two bandpass modulation schemes are no longer available (*e.g.*, when N equals 0.5dB, CCK at 11Mbps yields the lowest $E_{data-send}$).

However, considering $E_{data-send}$ with only P_t, always choosing the fastest bandpass modulation scheme is also not the most energy efficient. When N equals 1.1dB, all four bandpass modulation schemes are available. However, CCK at 5.5Mbps yields the lowest $E_{data-send}$, not CCK at 11Mbps (the fastest) or DBPSK (the slowest). This seems to indicate that a complicated scheme such as Miser [40] is required to choose the most efficient bandpass modulation scheme, transmit power level combination. However, $E_{data-send}$ does not represent the total energy consumption for transmission.

3.3.3 Total Communication Energy

To show that fast data transmission produces the most energy efficient communication for wireless LANs, it is necessary to look at the total energy consumption for transmission, $E_{data-send}$, for each bandpass modulation scheme. Recall that $E_{data-send}$ factors in P_{xmit} and P_{base} (see Equation 3.3). To factor out the energy needed to keep the card idle and ignore the effect of power management, let $P_{base} = 0$ and $P_{xmit} = 980mW$. Assuming $D = 1$Kb, then $E_{data-send}$ for each bandpass modulation

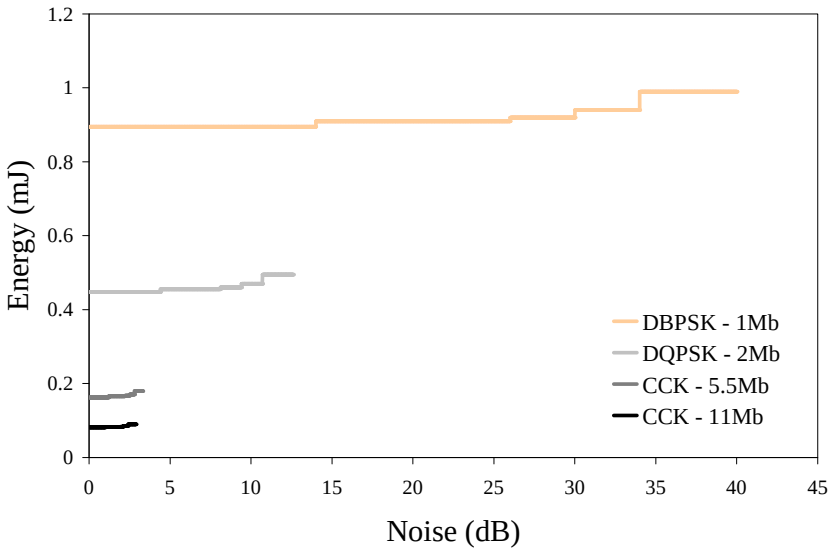

Figure 3.4: Total Communication Energy Consumption

scheme is represented by the step functions in Figure 3.4. The x-axis represents N in dB and the y-axis represents $E_{data-send}$ in mJ. Again, the most energy efficient strategy chooses the bandpass modulation scheme, transmit power level with the lowest $E_{data-send}$. As Figure 3.4 depicts, once $E_{data-send}$ is considered, even with $P_{base} = 0$, the fastest rate bandpass modulation scheme always has the lowest $E_{data-send}$ and therefore is the most energy efficient.

A power management scheme would increase the energy savings achieved by using the fastest rate bandpass modulation scheme. To see this, consider a perfect power management scheme that places the card in sleep mode whenever it is not transmitting. This essentially allows P_{base} to be factored into $E_{data-send}$ (see Equation 3.3), yielding $P_{base} = 1.275$mW, $P_{xmit} = 980$mW, and $D = 1$Kb. It is clear that, factoring in P_{base}, the gaps between the energy consumptions of the bandpass modulation schemes must grown, making the fastest bandpass modulation scheme clearly the most energy efficient.

This analysis of total energy costs leads to the consideration of the point where fast data transmission is no longer efficient. The gap between transmit and idle drives the fact that faster data transmission is always better. If this gap becomes much less, faster may not always be better. However, cards would have to become much better for that to be the case and radio technology is a mature discipline. RF transmitters are not becoming rapidly more efficient. Furthermore, there is a tradeoff between device size and efficiency [51]. The smaller RF devices become, the less efficient

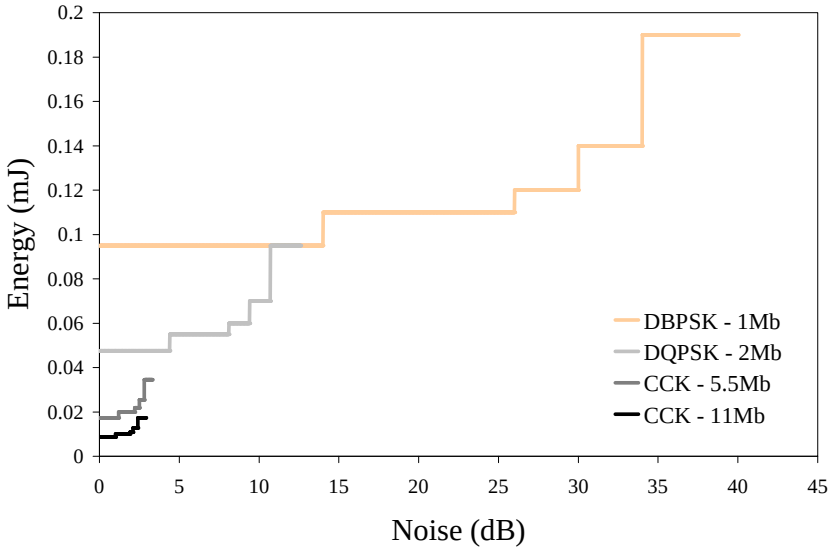

Figure 3.5: Efficient Card Energy Consumption

they are. This is a real issue for mobile devices since there is an obvious push to reduce the size of the devices. Therefore, P_{base} and P_{xmit} are not going to become negligible in the near future. Therefore, faster data transmission (*i.e.*, keeping the time spent in transmit mode to a minimum), will continue to lead to energy efficient communication.

To demonstrate that cards would have to become radically more efficient, we find the minimum value of P_{xmit} for which the fastest bandpass modulation scheme is always the most efficient. This value essentially shows how much more efficient RF circuitry needs to become before fast data transmission fails to be the most efficient. Let $P_{base} = 0$ and $P_{xmit} = 90$mW. E_{total} for each bandpass modulation scheme is represented in Figure 3.5. E_{total} for DQPSK and DBPSK converge for N_r around 12dB. Therefore, if P_{xmit} were reduced further, DBPSK would have lower E_{total} than DQPSK at 12dB and the fastest rate would not always be the most power efficient. However, 90mW is less than the highest transmit power level. In fact, this value is less than the hardware MAC processing costs and almost two orders of magnitude less than the cost of today's RF amplifiers [11]. Therefore, the faster-is-better approach is not likely to be outdated in the near future.

3.4 Pincher

To yield the most effective and energy efficient communication given the lossy nature of wireless communication and the power-constrained nature of mobile computing devices, we present an approach that jointly adapts the bandpass modulation scheme and the transmit power level. Our approach also considers the impact on any savings from wireless interface costs (*i.e.*, the cost of keeping the interface in transmit mode). In this section, we present Pincher, a wireless communication protocol that achieves energy efficiency through fast data transmission. At a high level, Pincher first uses information about the SNR at the receiver to determine the bandpass modulation scheme with the fastest possible rate. It then uses TPC to match the SNR at the receiver to the threshold SNR of the chosen bandpass modulation scheme. By choosing the fastest data rate and minimizing the time spent in transmit mode, Pincher complements idle-time power management. Finally, we present the results of an ns2 simulation to add further verification to the results derived from the analytic model and the prototype measurements.

3.4.1 Adapting the Bandpass Modulation Scheme

The choice of bandpass modulation scheme must balance three tradeoffs. First, the higher the transmission rate of the bandpass modulation scheme, the lower the loss tolerance. Second, the higher the transmission rate, the lower the time to transmit each application bit. Third, the higher the transmission rate, the less time the interface must be in transmit mode, increasing the benefit of power management. Given these considerations, Pincher uses the highest transmission rate bandpass modulation scheme that can tolerate the current error conditions in the signal, minimizing the energy consumption for the transmission stream.

Pincher bases its choice of bandpass modulation scheme on the receiver indicated channel quality sent during channel negotiations (*i.e.*, RTS/CTS in IEEE 802.11). Each bandpass modulation scheme has a SNR_t for which it can maintain communication. We measured the performance of a Cisco Aironet 350 series card to find the values of SNR_t for each of the four bandpass modulation schemes supported (*i.e.*, 1Mbps, 2Mbps, 5.5Mbps, and 11Mbps).

To choose a bandpass modulation scheme, Pincher merely compares the current SNR perceived by the receiver (SNR_r) to the values in the table and uses the bandpass modulation scheme with

the highest R for which the SNR is above the SNR_t.

3.4.2 Choosing the Transmit Power Level

Once a bandpass modulation scheme is chosen and SNR_t is established, P_t can be lowered to achieve SNR_t given the current noise level at the receiver. The bandpass modulation scheme is determined based on the signal strength at the highest P_t to provide the most accurate measurement of the maximum possible SNR [22]. Since the choice of bandpass modulation schemes is relatively coarse-grained, it is likely that the SNR at the receiver is above SNR_t for the chosen bandpass modulation scheme. Therefore, Pincher uses TPC to reduce P_t to achieve SNR_t.

Actual, current wireless interfaces do not have continuous power control. To support such limitations, Pincher uses discrete power levels to find thresholds within each bandpass modulation scheme, yielding a table indexed on noise containing bandpass modulation scheme / transmit power level pairs.

To summarize, when a packet is ready to be sent, according to IEEE 802.11, the sender transmits an RTS packet at the highest P_t. The receiver responds with a CTS containing SNR_r sent at the highest P_t. The sender uses SNR_r to perform a table lookup to choose the appropriate bandpass modulation and P_t. The data is sent by the sender using the chosen bandpass modulation scheme and P_t. Finally, the receiver responds with an acknowledgment sent with P_t.

3.4.3 Implementation

To demonstrate that Pincher can be easily built into a system, we have implemented a prototype of Pincher on an Dell Latitude laptop with 256MB RAM. The OS is Red Hat Linux 9.0 with kernel 2.4.20. To gain benefits from TPC, an interface card that implements various transmit power levels is required. Such a card must also provide a means to switch between these transmit power levels, as well as switching between bandpass modulation schemes without significant delays. Unfortunately, the cards on the market require a hard reset each time either type of change is made. Therefore, our evaluation emulates the type of card we expect to see in the future that smoothly and efficiently supports BPM and TPC. In our emulation, no switching time between transmit power levels or bandpass modulation schemes occurs. Additionally, we assume that the difference

in energy consumption from TPC is reflected in P_t. For example, when switching from a 100mW P_t to a 50mW P_t, the interface card consumes 50mW less. Finally, to take advantage of energy savings from putting the card into sleep mode, a power management scheme must be available. In this thesis, no such scheme is assumed. Therefore, the current evaluation only discusses benefits from the power savings achieved by considering the costs driving the transmitter. Because Pincher uses fast data transmission, if a power management scheme is used along with Pincher, the energy savings will increase.

Implementing Pincher is very straightforward. IEEE 802.11h [25] supports the return of SNR_t to the sender in CTS packets. The correct bandpass modulation scheme and P_t are selected according to the table lookup as described in Section 3.4. Finally, wireless interface driver calls are used to set the bandpass modulation scheme and the transmit power level of the wireless interface.

One final design decision is the number of MAC-layer retransmissions to attempt before giving up on a packet. Chockalingham, *et. al.* [6], show that link-layer retransmissions must continue long enough to outlast the average bad state duration in order to yield benefit over a protocol not using retransmissions. To this end, common IEEE 802.11 implementations use 7 data retransmits [40] before giving up. Pincher uses the same threshold.

3.4.4 Simulation Results

Using the implementation of Pincher, we located through experimentation the noise thresholds for which the bandpass modulation scheme, TPC settings must be changed. To verify the results of the analysis in a dynamic environment, we simulated Pincher using the ns2 network simulator [34], using the thresholds found with the prototype implementation as the thresholds for the simulation. We started with two nodes at the maximum transmission distance from each other using the 1Mbps bandpass modulation scheme and the 5mW P_t. For these simulations, we use a very small 90mW value for P_{xmit} and 0mW for P_{base} as in the experiment used in Figure 3.5 to model a highly efficient interface card.

Figure 3.6 depicts the results for the ns2 simulation. At the beginning of each simulation, the sending node begins moving toward the receiving node at either 1, 5, 10, or 15 m/s. The x-axis represents the number of packets sent. The y-axis represents the cumulative energy expended to

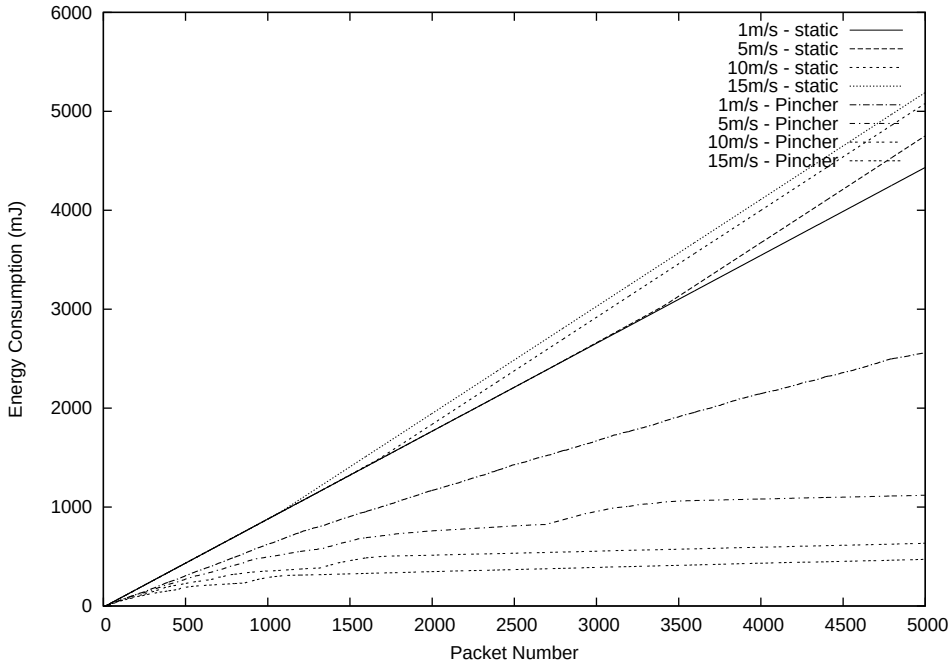

Figure 3.6: Pincher vs. Slowest Rate Protocol

send the data.

The static configurations use 1Mbps with 5mW transmit power level for the entire run. The four lines on the bottom represent Pincher runs, and the four top lines are the static runs. As expected, the more rapidly the sending node is able to speed up its transmission, the cheaper the cost in terms of energy expended.

3.5 Contributions and Future Directions

The contribution of this section is the design and implementation of Pincher, an energy-efficient MAC layer protocol using fast data transmission.

Future work includes extending the implementation of Pincher to cover IEEE 802.11a/g. Currently, Pincher is only implemented for IEEE 802.11b. Because BPM dominates power consumption, extending Pincher to MAC layers with higher data rates should increase the energy-efficiency gains. Also, different cards have costs associated with changing between bandpass modulation schemes. These costs are in terms of energy spikes as well as delays in active times in functionality. Factoring in these costs changes the optimal times to switch bandpass modulations or P_t.

Chapter 4

Adaptive Transport Layer

The transport layer provides a network interface directly to the applications with a goal of providing the network services needed by the applications with a suitably friendly application interface. In the context of energy efficient mobile system design, the transport protocol has three requirements. First it must be designed to support varied application requirements. Bulk data transfer applications require that 100% of the data being transferred arrives at the receiver. However, such applications rarely have strict timing requirements for the arrival times of various fragments of the data. Multimedia applications, in contrast, typically have strict timing constraints on the arrival of individual frames of data (*i.e.*, frame playout times). These applications can sustain some loss in the stream while still maintaining acceptable functionality. Second, the transport protocol is responsible for minimizing energy consumption within the constraints of supporting the applications. Finally, the transport layer must export information that is required by adaptations in other layers to support the cross-layer adaptation model.

To achieve the goal of supporting application requirements, a number of mechanisms have been studied, from ensuring timeliness [35], to performing congestion control [48], and providing variable reliability levels [30]. However, none of these mechanisms, when faced with energy constraints, deals completely with the goals of minimizing energy consumption while supporting varied application needs. Furthermore, many traditional mechanisms that perform optimally in the absence of energy constraints often perform poorly when confronted with such energy constraints [55]. In this chapter we focus on the issues relating to multimedia applications because of their strict timing constraints and ability to sustain some loss. The protocol presented in this chapter can also support the full reliability requirements of bulk data applications.

An understanding of the causes of energy consumption at the transport layer facilitates the ability to address the support of dynamic application requirements in the face of energy constraints. Essentially, the transport layer consumes energy in two ways. First, the protocol itself will consume energy due to its execution on the CPU. The more complex the algorithms used by the transport layer, the more CPU energy it will consume. Therefore, not only for time efficiency but also for energy efficiency, it is necessary to maintain minimal complexity in the transport layer algorithms.

The main transport layer energy consumption, however, comes from the phenomenon of data expansion. The transport layer provides its services by adding data to the application data frames. This additional data costs the system in terms of the amount of energy to transmit the data as well as in terms of the amount of time it takes to transmit the data. Essentially, data expansions effects can be thought of in terms of the average number of bytes required to send one application byte. As this number increases, the efficiency of the transport layer decreases. Data expansion comes from two sources. First, it is often necessary to add transport layer headers to each transport layer fragment. If an application data frame is fragmented by the transport layer, then each fragment will contain a transport layer header that will add size to the total number of bytes required to perform the transfer. Second, loss recovery causes extra data to be sent across the network to recover lost data. This is the largest cause of data expansion. The triggers for loss recovery drive the data expansion. Therefore, the fundamental question that must be answered before an energy efficient transport protocol can be designed is: *What causes frame loss?*

A frame is lost when it is not usable by the receiving application for some reason. There are essentially two reasons why a frame may become unusable to the application. First, a frame could arrive at a time past its playout time. Second, a frame could arrive incompletely, due to a loss of one or more of its fragments. In the case of a late frame, there is no repair option, if the frame is late at the time it is received, any repair attempt will also be late. Therefore, our approach to dealing with frame loss due to lateness is to detect losses and attempt to avoid them. In the case of a frame arriving incompletely, it is possible to repair the frame, so long as its playout deadline is not violated. Therefore, our approach to dealing with this case involves detecting a fragment loss and attempting to repair the frame so long as we don't violate its playout deadline by doing so.

Recall that multimedia applications often times encode frames before sending them to reduce

the size of data from an initial raw size D to a reduced size D'. The details of this data reduction are given in Chapter 6. Once the application has encoded a frame, it is the job of the transport layer to send that frame. Since it must at least add headers to the data, the transport layer always imposes some overhead, and D'' bytes are actually transmitted per D' application bytes. $\frac{D''}{D'}$ represents the expansion factor for the transport protocol. Therefore, the energy to transmit the D' bytes is defined by this expansion factor, the transmission rate of the interface card (R), and the amount of power to keep the interface card transmitting ($P_{card} = P_t + P_{xmit} + P_{base}$):

$$E_{data-send} = \frac{D''}{R} \times P_{card}.$$

Although multimedia applications can often tolerate some loss, excessive losses may detrimentally affect the quality of the stream. Additionally, since the application has already consumed some amount of energy to encode the frame, not recovering from loss may waste that energy. To recover from loss, the transport protocol can add redundancy into the data stream, increasing D'', and so increasing $E_{data-send}$. Again, although the application does not want to know the details of loss recovery, the impact of loss recovery on per-application-byte energy costs must be exposed.

4.1 On the Tolerance of Losses

One of the primary components of a network service is to provide sufficient loss recovery mechanisms to support the needs of the applications. Such loss recovery is achieved by increasing the number of bytes transmitted, either by adding redundant data to allow the receiver to reconstruct the original application data itself, as in the case of forward error correction (FEC), or more directly by retransmitting lost application data. However, loss recovery comes at the cost of increasing the energy consumed to transfer the application data. The additional energy investment necessary for reliability creates a tradeoff between perfect reliability and energy efficiency. However, these traditional loss recovery mechanisms that perform optimally in the absence of energy constraints perform poorly when confronted with such energy constraints [55]. We therefore evaluate these mechanisms to determine their impact on energy consumption.

Since application data units (*i.e.*, frames) are often much larger than the maximum packet

size of the network, a transport protocol typically divides them into fragments. These fragments are then transferred across the network and reassembled before delivery to the application. Each packet that is lost in the network carries one of these fragments, and so fragments are the unit used for loss recovery. For a multimedia frame to be useful to the receiving application, all of its fragments must be received and reassembled before the frame's deadline. Since perfect loss recovery is expensive, and is usually unnecessary in the case of multimedia data, suspending loss recovery in certain cases can save energy and bandwidth. In this section, we first present two typical loss recovery mechanisms, FEC and retransmissions, and evaluate their effect on energy consumption and reliability in the face of timing constraints. Given that neither approach can provide energy-efficient 100% reliability and that multimedia applications may not require such reliability, we present two scenarios when loss recovery should be suspended and the subsequent impact on energy consumption.

4.1.1 Loss Recovery

As a proactive approach, FEC achieves loss recovery by adding some small amount of redundancy to each fragment of the transmitted data. In this case, $D'' = D' \times (1 + \mu)$, where μ is the amount of redundancy added per frame and so defines a correcting threshold. Since creating the error correcting codes requires computation and so incurs an energy cost, E_{comp}, the total energy cost for FEC-based loss recovery is

$$E_{FEC} = \frac{D'(1+\mu)}{R} \times P_{card} + E_{comp}.$$

FEC can only recover if the number of lost fragments remains below the correcting threshold. The larger the value of μ, the larger the correcting threshold and the more lost fragments can be recovered. The main benefit of such proactive approaches is immediate data recovery at the receiver. In the face of a loss, no time is wasted on retransmissions. The cost of FEC comes from the extra $\mu D'$ bytes added to each frame and from E_{comp}. If the number of lost fragments is below the correcting threshold, bandwidth and energy are wasted in the transmission of the extra redundancy. On the other hand, if μ is too small, loss recovery will fail and backup mechanisms will have to be used to repair the data stream, increasing E_{FEC}.

Retransmission-based loss recovery reactively recovers from loss by retransmitting lost fragments only when a loss is detected. Such approaches have the advantage of increasing the number of bytes transmitted, and so consuming extra energy, only when there is a loss. Therefore,

$$D'' = D' \left(1 + \frac{k}{K}\right),$$

where K is the number of fragments and n is the number of retransmissions of fragments needed to recover the data. Since there is little computational overhead for retransmitting fragments, the total energy cost for retransmission-based loss recovery is

$$E_{retrans} = \frac{D'(1 + \frac{k}{K})}{R} \times P_{card}.$$

However, since losses typically cannot be detected before a round trip time has elapsed, it is possible that any retransmissions will arrive too late to repair the data, wasting bandwidth and energy on the unusable retransmissions. While it is obvious that retransmission-based recovery can cause the frame being repaired to be late, such techniques can also make subsequent frames late, wasting even more energy.

To determine which method is most energy efficient, it is necessary to compare the expected energy costs for the target environment. When the FEC code matches the loss rate of the channel exactly, μ is equivalent to $\frac{k}{K}$, since all of the redundancy added by the FEC is used for repair. Even in this case, however, the overhead of E_{comp} makes FEC more expensive in terms of energy than a retransmission-based strategy. Additionally, in bursty environments like the Internet and wireless networks, the loss fraction $\frac{k}{K}$ varies, making it impossible to match the FEC error correcting ability to the loss rate. This results in either extra redundancy or overhead from using other repair techniques like retransmissions. By overestimating the loss rate, FEC can work well in wired environments (either in conjunction with application frame encoding techniques [53] or in multicast environment [36, 44]) where energy constraints are not an issue. In such environments, the rapid recovery outweighs the extra overhead for unused redundancy, which has little impact on performance. However, in wireless environments where energy is a major constraint and network conditions vary rapidly, the overhead for FEC is too high.

While FEC-based loss recovery is not effective in energy-constrained applications, neither is it the case that retransmission-based loss recovery is a perfect solution. When a retransmission arrives at the receiver in a timely fashion, the fragment is still beneficial to the application and so the energy overhead of the retransmission was not wasted. However, if the retransmission arrives too late or was not necessary, the energy put into the retransmission was wasted. This begs the questions of when, if ever, retransmissions should be attempted.

4.1.2 When a Frame Isn't Worth Saving

There are two unique characteristics of multimedia data that make retransmission-based loss recovery challenging. First, since multimedia applications have strict timing requirements, any frames that arrive past their deadlines waste energy. Early identification of these frames can save both energy and bandwidth. Second, the ability to tolerate some threshold of loss opens the possibility of opportunistically suspending loss recovery to save energy. In this section, we discuss both of these characteristics and their impact on energy consumption. In the following section, we show that while each of these approaches in isolation can save some minimal amount of energy, the benefits of combining them is greater than the sum of the individual improvements.

Timing Violations

Arriving frames that violate timing constraints cannot be used by the application and so all the energy that went into encoding and transmitting those frames is wasted. Although it is not possible to entirely eliminate such waste, two approaches can be used to reduce it. *Reactive* methods let the receiver notify the sender when a frame or a fragment of a frame is received late. The sender can then avoid sending any further fragments of that frame [46]. *Predictive* methods allow the sender to predict if a frame or a fragment of a frame will be late. The sender can then avoid sending anything that is predicted to be late. An analysis and design of this method is one of the contributions of our work.

Reactive methods leverage the fact that the receiver knows the deadline of a frame. Therefore, the receiver can tell if that deadline has passed during the reception of the fragments of a frame. The receiver can then signal the sender to stop sending any further fragments of that frame [46].

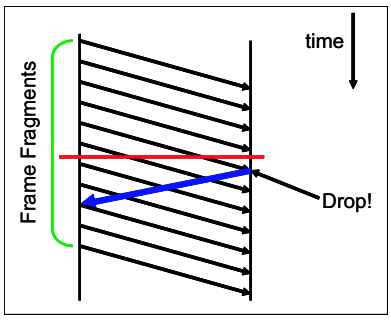

Figure 4.1: Receiver Late Frame Mechanism

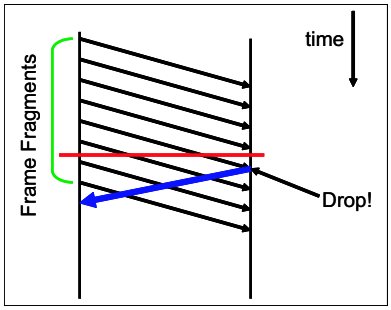

Figure 4.2: Receiver Late Frame Mechanism Failure

For example, the last two frames in Figure 4.1 could be dropped at the sender. While this can save the transmission energy of the unsent fragments, the energy to encode the frame and the energy used to transmit the initial fragments is wasted. Additionally, the ability of this mechanism to save energy by preventing the sender from transmitting useless fragments of a frame depends on the size of the frames, the bandwidth, and latency of the link. If all of the fragments of a frame are in flight before the drop message is received by the sender, this mechanism results in no energy savings (see Figure 4.2).

Predictive methods leverage the fact that the sender has some information about frame deadlines and estimations of the round trip time (RTT). Therefore, the sender can predict the one-way latency and so predict whether a frame will arrive late. Similar to the reactive methods described above, such predictive methods can save the energy from transmitting unsent fragments. Additionally, if the prediction is made early enough, it may be possible to signal the application not to encode the frame, hence saving all energy associated with the frame. The challenge to predictive methods is that they require accurate estimations of network latency. If the estimation algorithm is too pessimistic, frames that could have arrived on time may get dropped, and if it is too optimistic,

frames that can never arrive on time may still get transmitted.

Both methods are limited by the relationship between the network latency and the frame deadlines, and so applications with timing constraints can be made more robust to timing violations by the use of a playout buffer at the receiver to increase the amount of time available for retransmissions [35]. We discuss the impact of such a playout buffer on loss recovery in Section 4.2.1.

Opportunistic Dropping

It is obvious that suspending loss recovery for frames that are already late or predicted to be late can save energy and bandwidth. However, it is also possible to leverage information about current performance (in terms of the fraction of frames successfully delivered) to decide whether to suspend loss recovery even if the frame might arrive on time, improving energy efficiency and successful frame delivery rate.

If the receiver is not experiencing many dropped frames, it is possible that the application could tolerate the loss of a frame. Given an application-specified reliability threshold, it may be possible to save energy by opportunistically dropping a frame with lost fragments and avoiding the retransmissions. This approach has two effects. First, energy that would have been expended in the retransmission of the lost fragments is saved. Second, the stream can catch up in time by skipping ahead to the next frame, potentially even preventing subsequent frames from being late. Additionally, since packet loss frequently indicates congestion in the network, it may be the case that a retransmission would be lost anyway, and that trying to force the frame through would simply be throwing good energy after bad.

If the receiver has been tracking the quality of the received data stream, the receiver may determine that the current quality is above the application's threshold. A smart receiver could then decided to not ask for retransmission of the lost fragments [30], saving energy from the retransmission [10]. This can be implemented by allowing the receiver to send a false acknowledgment of lost fragments when the quality is higher than necessary. We take this approach one step further and allow the receiver to send a false acknowledgment for the whole frame, allowing the sender to avoid sending any unsent fragments in the frame.

In the next section, we discuss how these techniques can combined synergistically to provide

superior on-time delivery rates while maximizing energy efficiency. We also present a prototype implementation. Compared to existing protocols our prototype provides higher delivery rates and expends less total energy to do so.

4.2 Reaper: An Energy-Aware Transport Protocol

Given our insights into loss recovery, we designed an energy-aware transport protocol, Reaper, that minimizes energy consumption while maintaining target application reliability requirements using a novel combination of loss recovery mechanisms. By exploiting minimal use of opportunistic dropping, Reaper can more effectively use predictive dropping to minimize the number of reactive drops and increase the application goodput. In this section, we discuss two of Reaper's parameters that control this use of opportunistic and predictive dropping. Along with intelligent dropping, Reaper also supports adaptive applications by exposing relevant information about network cost and availability. Additionally, Reaper optimizes energy consumption using simple application-specific information about data reliability requirements and timeliness. To evaluate the effectiveness of Reaper and its cross-layer design, we implemented and tested a real-world implementation. Our evaluation of this prototype shows the benefits from both Reaper's effective energy conservation techniques and the exchange of cross-layer information. Our results demonstrate the need for such cross-layer information in protocols.

4.2.1 Loss Recovery Policies

To optimize the fraction of frames that successfully arrive at the receiver while minimizing the unnecessary expenditure of energy, Reaper employs intelligent dropping policies that minimize timing violations and take advantage of opportunistic dropping.

Reactive Dropping

When a fragment of a frame is received past the frame playout time, the receiver signals the sender to drop any unsent fragments remaining in the frame. This signal can be included in an ACK sent by the receiver to minimize overhead. Because reactive drops cannot be triggered until the receiver detects a timing violation, many late fragments will have already been sent, wasting energy. If the

detection is too late, the sender will already have sent all fragments, resulting in no energy savings. Therefore, while reactive dropping can be used as a fallback, avoiding reactive drops altogether can save even more energy. Additionally, since the time has already been wasted to send the frame, reactive drops do not help the stream catch up in time, and so may not avoid further late frames.

Predictive Dropping

When a sender predicts that a frame being sent may be late at the receiver, the sender drops any unsent fragments of that frame. Such predictions must be made based on information about the deadline of the frame, as indicated by the application. The specific deadline for a frame depends on the frame rate of the multimedia stream, the inter-frame spacing IFS, and the size of the playout buffer at the receiver, B_p, in seconds.

For each fragment of a frame, it is necessary to estimate its frame's playout time and the estimated arrival time of the last fragment of its frame. To this end, the sender needs the information both from the application and about the current network conditions. From the application, the sender requires B_p, IFS, and the frame number for each fragment. The sender also requires a continuously updated estimate of the network latency L, which can be is estimated from the current roundtrip time ($\overline{RTT}$) as $\frac{\overline{RTT}}{2}$. Because the transmission time is negligible compared to the network latency, an accurate estimation does not require an estimate of the network latency that separates out the full bandwidth component of the roundtrip time.

To estimate the timeliness of a frame it is necessary to predict if its last fragment will arrive on time. If a frame fits in a single packet, then it is only necessary to determine whether the packet containing the frame will arrive on time at the receiver. For a given frame F_i this is equivalent to determining whether the following inequality holds:

$$t + \frac{\overline{RTT}}{2} < t_s + IFS \times i + B_p,$$

where t is the current time and t_s is the time at which the receiver began playout.

However, in practice, video frames are split into multiple fragments, and simple network latency cannot be used to make this lateness prediction. The determination of whether a fragment will be useful depends not on its arrival time, but on the arrival time of the last fragment of its frame. If

there are f fragments left to be sent in frame F_i, then for the last fragment to arrive on time, the following inequality must hold:

$$t + (T_f + \frac{\overline{RTT}}{2}) < t_s + IFS \times i + B_p,$$

where T_f is the amount of time to transmit the remaining fragments of the frame.

In the best case, T_f is negligible. In the worst case, each of the $f - 1$ fragments after the current one will take as much time to send as the current fragment. These edge cases represent the endpoints of the prediction interval

$$\left[\frac{\overline{RTT}}{2}, f \times \frac{\overline{RTT}}{2}\right].$$

Predictions based on the left-hand side of this interval will be optimistic and may cause fragments to be sent that will arrive late, while those based on the right-hand side will be pessimistic, and may cause frames to be dropped that would have arrived on time. To control the optimism of Reaper's prediction, we provide a parameter ω to shift the prediction within the interval. Setting $\omega = 0$ corresponds to the left endpoint of given interval, and setting $\omega = 1$ corresponds to the right endpoint. Section 4.2.4 presents an evaluation of the effects of the choice of ω on Reaper's performance in terms of the total number of dropped frames.

Thus, estimating whether a frame F_i will arrive on time is equivalent to checking whether the following inequality holds.

$$\begin{aligned}(i \times IFS) - (t - t_s) + B_p \ > \\ ((\omega \times \overline{RTT}/2 \times f) + ((1 - \omega) \times \overline{RTT}/2).\end{aligned}$$

Finally, the sender must estimate the start time of the video playout. To estimate this start time, Reaper must calculate which frame fills the playout buffer and so triggers the stream to start playing. This frame, F_s, can be calculated as follows:

$$F_s \ : \ s \geq \left\lceil \frac{B_p}{IFS} \right\rceil.$$

Given F_s, Reaper can predict t_s based on an estimation of the arrival time of F_s. Any frame sent before the stream starts will arrive on time. Once frame F_s has been sent, Reaper can start predicting whether subsequent frames will be late.

The largest gains in performance, in terms of good frames delivered, are due to predictive dropping. Predictive drops both conserve the energy to send frames that would have been late and allow subsequent frames to be sent sooner, building back some playout buffer at the receiver. Extensive evaluation is given in Section 4.2.4.

Opportunistic Dropping

Even if the sender estimates that there is enough time to recover a lost fragment, it may not be beneficial, in terms of both energy consumption and average loss rate, to try to recover the frame. This decision can be based on network specific information, such as knowledge about the current level of congestion in the network, or based on application specific information, such as knowledge about the reliability requirements of the application.

To support application reliability requirements, Reaper provides a top threshold (Γ). If the current reliability level is above Γ, lost fragments are never retransmitted and the associated frames are dropped. Disabling such retransmissions has two effects. First, dropping frames that are not necessary to send instead of repairing them through retransmission conserves the energy that would have been spent on the retransmissions. This is the primary function of Γ. The secondary effect of disabling retransmissions is that each frame that is dropped allows the next frame to be sent sooner, potentially preventing that frame from being late as well.

Although the opportunistic drops waste the energy from encoding the frame and transmitting the initial fragments, they frequently prevent energy loss from subsequent frames that would have otherwise been late. Additionally, opportunistic drops allow a stream with very few frames left in its playout buffer to build back up a reserve, preventing large skips in the stream, and reducing the number of predictive and reactive drops needed later.

4.2.2 Information Sharing Between Layers

Given our energy analysis of the application and network, a small amount of information must be passed between layers to facilitate cross-layer adaptation. This information can be divided into two categories: information that is passed to the transport layer from the application, and information passed to the application from the transport layer. Since Reaper optimizes data transmission based on application information, it is necessary to understand which information needs to be passed between the layers.

From the application, Reaper needs information about frame deadlines and the playout buffer to support predictive dropping. Real-Time Protocol (RTP) [49] can be used to support similar functionality, without interfering with the reliability mechanisms. If the multimedia streams have heterogeneous timing and reliability needs, more sophisticated techniques [32] can be used. Additionally, to support opportunistic dropping, Reaper needs information about the reliability requirements of the application. If the application does not provide a reliability level, Reaper defaults to 100% reliability so that applications that are not Reaper-aware can still function.

Information passed to the application from Reaper represents information about the cost and quality of the data stream. Reaper passes information about the current average number of bytes per good byte sent ($\frac{D''}{D'}$). This, combined with information exposed by the MAC layer about interface costs, allows the application to estimate the cost in terms of energy for sending data. Reaper also informs the application of frame drops so that it can react to the loss of frames if needed. For example, if the encoder is using a predictive method of encoding, the loss of one frame may imply the loss of following frames that are already encoded based on that frame. Therefore, the sooner the encoder is notified of frame loss, the fewer frames will be lost.

4.2.3 Prototype

To validate our design, determine the effect of the various parameters, and evaluate Reaper, we implemented Reaper as a user-space library under Linux. The library implementation of Reaper takes data from the applications that links to it and sends it over UDP, with the timing and reliability described as above. A final implementation of Reaper would, of course, be done at the kernel level to take advantage of the decreased packet overhead and system latency.

Reaper uses the TCP New Reno congestion control mechanisms [48], ensuring TCP-friendly behavior. However, the decisions about what data to send and when to retransmit a packet are based on the mechanisms described above. When retransmissions are used, Reaper uses the TCP New Reno's retransmission mechanisms, including timeouts based on estimates of RTT, and the Fast Recovery / Fast Retransmit mechanisms. Since these are well studied mechanisms, we do not go into further detail here. For future work, it would be interesting to develop a rate-based version of Reaper, evaluating the interactions of rate-based mechanisms with the mechanisms presented in Section 4.1.

4.2.4 Evaluation

Reaper uses a number of mechanisms to allow it the flexibility to conserve energy while still meeting the needs of the multimedia applications it is built to support. To demonstrate the effectiveness of using these mechanisms together in a single protocol, we use two metrics. First, it is important that the transport layer deliver enough frames to match the application's loss tolerance level. To measure this, we use the concept of application goodput, the number of good frames received on time by the application. Second, the data expansion caused by the transport layer must be kept to a minimum to increase energy efficiency. Therefore, we use data expansion as our energy efficiency metric. Using these metrics, we show that a complete, coordinated combination of the mechanisms presented above results in gains in both energy efficiency and the application goodput over using other subsets of the mechanisms. The greatest gains come from predictively dropping late frames, when combined with opportunistic drops.

For the experiments in this section, our testbed consisted of a Linux laptop running our implementation of Reaper sending data via IEEE 802.11b through a base-station to a wired node one hop away. The application simply sends unencoded data from a file to isolate the effects of using application level information at the transport layer. In Section 6, we present our adaptive application and show that further energy savings can be achieved by using bidirectional cross-layer information to also adapt the application.

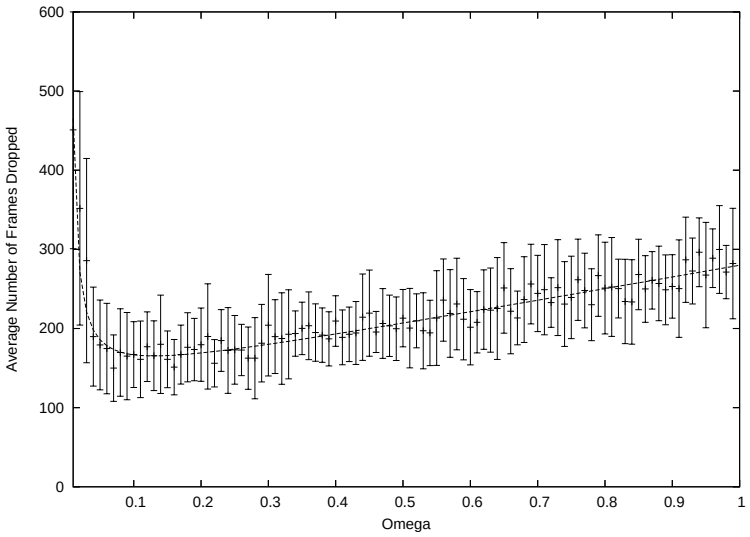

Figure 4.3: Frame drops as a function of ω, with the mean over 50 trials $\pm$ standard deviation.

Effects of the Parameters

Reaper has two parameters that can be used to alter the behavior of the protocol. This subsection analyzes the effects of each one in turn.

To support predictive dropping, ω affects how optimistically Reaper predicts that a frame will arrive on time. If Reaper is over-optimistic, it transmits frames that end up being late at the receiver. This aggressiveness actually results in the successful transmission of more frames because it drops fewer frames that would have actually gotten through. Being over-optimistic, however, results in a *decreased* energy efficiency because the number of frames sent that are late outweighs the extra frames that are pushed through, resulting in a much higher data expansion for the stream. Being over-pessimistic, on the other hand, reduces the number of frames that are actually late to near zero, but causes frames that could have arrived before their deadlines to be dropped. If many of these frames are partially sent before they are dropped, this pessimism can also cause a decrease in energy efficiency. If the frames are all dropped at the beginning of the frame, energy efficiency increases, but the total number of frames received, and therefore the user experience, suffers.

To evaluate the effect of ω on Reaper, we ran a large number of experiments on our real system (see Figure 4.3). These experiments confirm our intuition and indicate a good operating point of ω. With high values of ω, Reaper is too optimistic about whether there is time to get a frame through. This has the effect of causing many reactive drops, and increases the total number of drops. If ω

is too low, on the other hand, Reaper is too pessimistic and predictively drops a large number of frames unnecessarily. We see from Figure 4.3 that the optimal value for ω should be approximately 0.13. Although Reaper is relatively insensitive to the exact value chosen, values between 0.05 and 0.3 all give a total number of drops within 10% of the optimal value. Choosing a value far from this range, however, can drastically increase the number of frames dropped. For our experiments, we use the optimal value of 0.13. Because these experiments were performed over a wide range of network conditions over the period of a number of days, we have confidence that the omega value chosen will perform well in a wide variety of network conditions.

The use of optimistic dropping is determined by the top threshold (Γ), the maximum reliability desired by the application. This can be seen as the goal reliability level for Reaper. If the reliability level increases above Γ, Reaper disables retransmission of lost packets, and drops the associated frame to conserve energy. Essentially, lower thresholds allow more frames to be dropped during times when the network is in a particularly good state. This has the effect of conserving energy on channels where the effective bandwidth is greater than needed by the multimedia application. The ability of Γ to reduce the number of predictive drops is shown in the next section.

Performance Analysis

To understand the impact of combining the mechanisms used by Reaper, the following analysis presents goodput and energy analysis for six protocols: Reaper, Reaper without opportunistic drops, TCP with reactive drops, TCP with reactive and opportunistic drops, early drop, and TCP. TCP yields full reliability with no concern for timing constraints. Early drop aborts frames if losses are encountered, but has no mechanism to account for late frames. TCP with late frame drop sends all frames reliably but drops frames if notified by the receiver that they will be late. The unreliable variant disables retransmissions when the fraction of frames successfully received is above Γ. Finally, Reaper uses all of the mechanisms described in the previous section, with either full or application-specified reliability. The following results are averages over 50 runs. Each run transfers a 50MB file over an IEEE 802.11b wireless card in a laptop through a base station to a wired receiver one hop away. The wireless conditions and the network load vary over time.

We first evaluate the application goodput for each protocol. As seen in Figure 4.4, TCP and

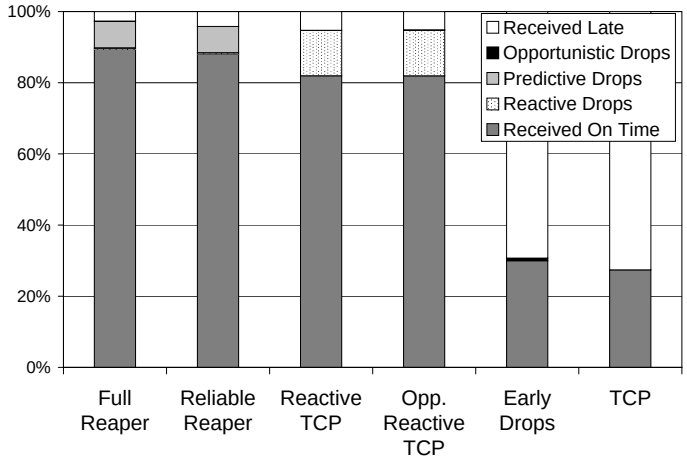

Figure 4.4: Frames Attempted and Their Disposition

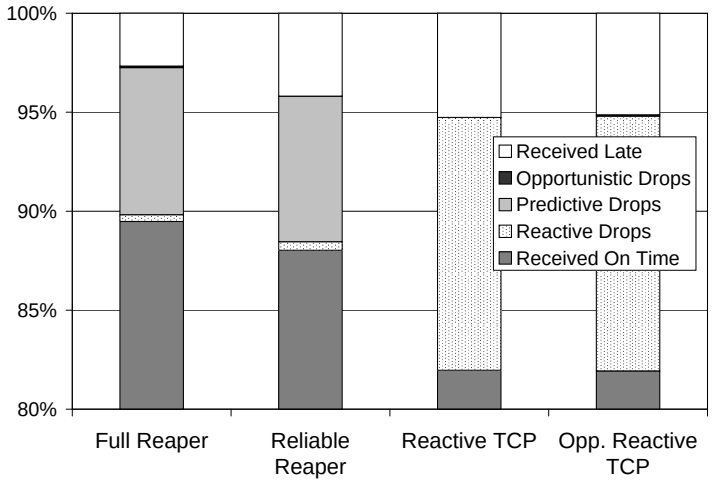

Figure 4.5: Magnification of Frames Attempted and Their Disposition

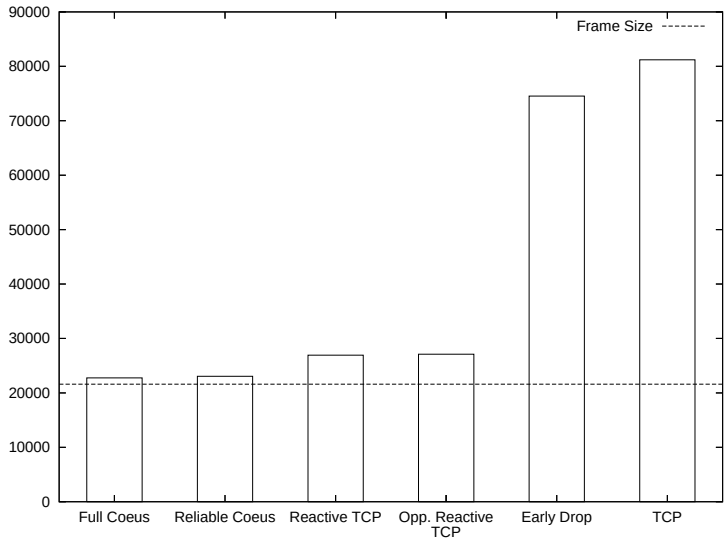

Figure 4.6: Bytes per Good Frame

early drop perform poorly. While TCP gets all of the frames to the receiver, less than 30% of the frames are on time. Early drop performs slightly better, due to its ability to use opportunistic dropping to catch up in the event of a packet loss. However, many frames are still received late. Reliable TCP with reactive dropping performs significantly better, getting roughly 82% of the frames through and on time. This improvement comes from the ability to abort sending frames that are arriving late. TCP with reactive and opportunistic drops performs slightly better (see Figure 4.5). Reaper's predictive dropping mechanism avoids nearly all reactive drops and increases the goodput nearly 10% over the opportunistic, reactive TCP. Reaper without opportunistic dropping gets approximately 88% of the frames through successfully. However, with opportunistic drops and Γ set to 98%, Reaper gets over 89% of the frames through to the receiver on time and intact (see Figure 4.5). The majority of the unsent frames are due to Reaper predicting that the frame will be late and skipping ahead, and thus saving the associated transmit energy.

Next, we evaluate the energy efficiency of the protocols. Recall that the energy consumption of a transport protocol is directly affected by the number of bytes used to transmit the data usable by the receiver. Essentially, this is the stream-level data expansion for the protocol. Figure 4.6 depicts the average bytes sent per good frame for each of the protocols. The line in the graph at about $\sim$22K bytes represents the average number of bytes per frame. Therefore the space above the line represents the average data expansion per frame. As expected, Reaper has a significantly

lower data expansion and therefore better energy efficiency compared to the other protocols.

In addition to sending significantly fewer bytes per good byte, Reaper actually sends at least 8% fewer total bytes than any of the other protocols, thereby using less total network energy. It is able to do this through its predictive dropping mechanism: since Reaper does not have to wait for the receiver to notify it about timing violations, Reaper can avoid sending even the first fragments of a frame which will be late.

However, as discussed in Section 2, the network is not the only, or even necessarily the dominant, component of multimedia system energy consumption. We examine the effect of application CPU energy consumption in the next section, and show how Reaper exposes the information necessary to build applications that can make intelligent decisions about the CPU/network energy tradeoff.

4.3 Contributions and Future Directions

The contribution of this section is the design and implementation of Reaper, an adaptive transport layer. Reaper provides mechanisms to allow it to be tailored to the needs of specific multimedia applications to provide sufficient stream quality while minimizing energy at the transport layer. Furthermore, Reaper supports exporting information for use by adaptations in other layers.

Future work includes considering whether an adaptive algorithm to dynamically chose frame threshold would help increase energy efficiency. Alternate means of gap detection, such as the use of selective acknowledgments could increase the efficiency of the protocol and its ability to repair more frames without violating playout constraints. Also, methods to increase the accuracy of sender late frame prediction could further increase Reaper's energy efficiency. Finally, implementing a rate-based version of the protocol and exploring the interactions of the mechanisms in that context would also be interesting.

Chapter 5

System Architecture

A final component of the system is the framework necessary for inter-layer information passing. this framework essentially defines interfaces for each mechanism in the system to post relevant information to the system as well as read information from other layers necessary for its adaptation.

To support diverse applications, it is important to present resource information at a level that is meaningful to applications. For example, an application that uses bandwidth information should not have to understand the configuration of the system or how other applications may impact bandwidth availability. To this end, the application needs a resource manager that exposes a simple interface, hiding the complex characteristics of the underlying system. One method information providers can use to expose information to consumers is to write such information to a location, which is often globally accessible, called a *blackboard* [23]. When an application requires some piece of information, it searches the blackboard. Our system architecture uses a resource blackboard that achieves this goal by providing an interface that is simple and usable by both applications and the operating system. The advantage of a blackboard design for our system architecture is the ease of exposing simple interfaces to each of the components of the system that need access to resource information (see Section 5.1).

To demonstrate the power of our approach, we have implemented a prototype of our system architecture, which we call Corvus, on the Linux platform. Through the use of multiple instances of a video streaming application over an IEEE 802.11b network interface, we are able to compare the energy consumption of Corvus to the performance of a regular Linux system and a system capable of supporting application priorities. These experiments highlight the benefits to the user when the system supports the use and distribution of system-level resources for adaptations.

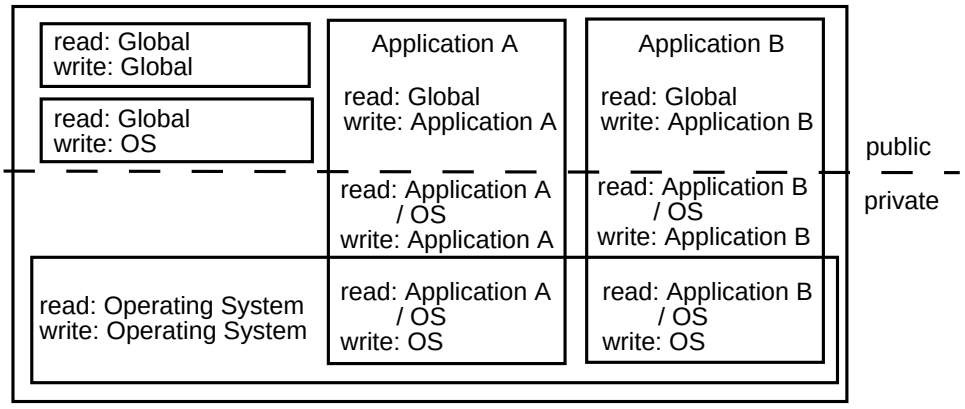

Figure 5.1: Resource Blackboard

5.1 System Blackboard

To provide a simple interface to expose resource information to both the resource manager as well as applications, Corvus uses a resource blackboard. In general, a blackboard provides a simple interface to resource information. Corvus extends the basic blackboard to support multiple resource information producers and consumers by providing each with its own section of the blackboard. Each section is controlled by the owner, allowing the easy integration of access control techniques. Figure 5.1 depicts the blackboard design used by Corvus.

The Corvus blackboard is divided into four types of areas. These four areas were designed to allow access control between system components. It may not be desirable for applications to be able to write over resource allocations written by the operating system. Also, each application should have a section that other applications cannot over-write. The first type supports the traditional use of globally readable / globally writable information. The second type is globally readable but only writable by the operating system. There is one instance of each of these first two types. The third type is for specific applications. There is one instance for each application. Each application's area is divided into three sections. The first section is writable only by the application but is globally readable. The second section is writable only by the application and readable by the application and the operating system only. The last section is readable by the application and the operating system but only writable by the operating system. The final type of area is only readable and writable by the operating system itself. This is the part of the blackboard used by the operating system to keep track of resource information it uses for itself (*e.g.*, interfaces available).

Corvus is designed to run on a single system to support adaptive applications and provide resource management. Since the number of sections on the blackboard grows with the number of active applications running on the single system, scalability is not a problem.

The resource blackboard gives Corvus the ability to serve as a resource manager as well as an information provider. The blackboard gives Corvus a clean interface with which to communicate to applications and other system modules. Since Corvus is a system-level service provider, applications gain the system-level support they need to effectively use system resource information.

While the blackboard provides a way for Corvus to tell each application how much of a resource it can use, policy enforcement modules are needed to prevent misbehaving applications from attempting to use more resources than Corvus allocates. For example, Corvus mandates the bandwidth each application may use by writing the bandwidth amount to the application section of the blackboard. However, if an application attempts to use more bandwidth, the blackboard has no mechanisms to stop this misbehavior. Such policing is left to a bandwidth enforcer. The design and implementation of the enforcers are the subject of future work.

5.2 The Corvus Prototype

The design of any system must be verified through implementation and use. The goal of our prototype of Corvus is to demonstrate the feasibility of making use of resource information at the system level while providing applications with a simple yet effective interface to Corvus. Internally to Corvus, our prototype provides a clean design for the representation and management of resource information and the resource blackboard. For ease of the initial implementation and testing, Corvus is implemented in user space.

To demonstrate our prototype, we implemented a video streaming application capable of adapting the frames per second. Corvus is used to manage the resources of two simultaneous instances of the application that together require more bandwidth than is available. The performance of Corvus is compared to a system with the standard Linux resource manager and a system with a priority quality of service manager. At this time, we assume that there are no misbehaving applications in the system.

5.3 Example

Consider a laptop equipped with an IEEE 802.11b interface communicating with two other computers connected to the Internet. Each of these fixed hosts is streaming an MPEG video to the mobile node at frame rates from 5 to 25 frames per second (fps). The laptop is equipped with a streaming MPEG player that can play MPEG videos with dynamically changing frame rates. The MPEG video player requests that the video be sent at higher or lower rates as the video is being played, allowing dynamic rate adaptation. Our experience with the video streaming applications determined that 20 fps are required for a good quality video.

To illustrate the benefits of using Corvus, consider the following situation. One of the videos is being streamed from a friend's house and the other from a colleague's desk at work. Since it is a work day, what is happening at the colleague's desk is more important. However, when the colleague is away, it is more desirable to be viewing the friend. Each webcam is running presence detection software capable of notifying when someone is present in the video field. Initially, only the stream from the friend is active. After 15 minutes, the colleague turns on the streaming application as well, so that both feeds are available. The colleague then gets up to get some coffee 10 minutes later without turning the video stream off, and then returns 10 minutes after leaving. While this scenario would be very straightforward if enough bandwidth were available, the IEEE 802.11b channel can only handle 30 fps in total.

Figures 5.2(a-c) show the amount of bandwidth used by each video stream in terms of frames per second. The light gray region represents the friend's video stream. The dark gray region represents the colleague's video stream. The height of the combined regions represents the total bandwidth used by both video streams.

First, consider a scenario using a standard Linux resource manager. As Figure 5.2(a) shows, when the first video stream is playing alone, it receives the full 25 fps. However, when the second stream becomes available, each stream averages 15 fps, causing both streams to be sent at an unacceptable quality. Nothing happens when the colleague gets up and returns since there is no system support for context about presence in the video field.

Second, consider a scenario using a priority quality of service scheme on the laptop. This type of scheme allows applications to be given priority, but does not support the use of context. As

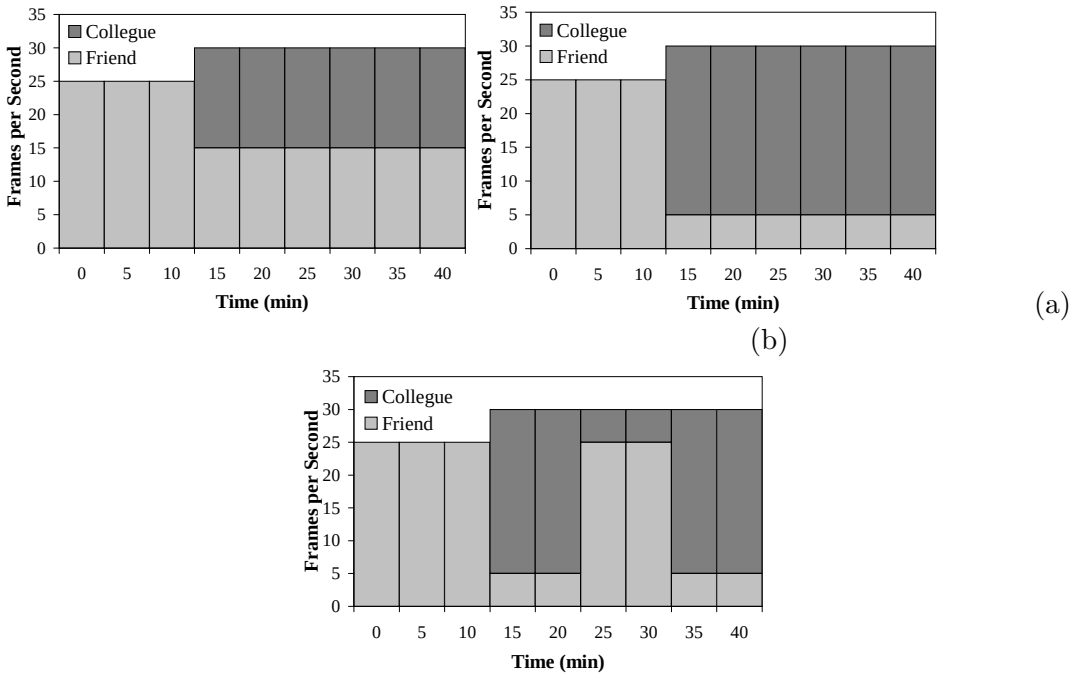

Figure 5.2: Results: (a) Standard Linux Manager, (b) Priority QoS Manager, (c) Corvus

Figure 5.2(b), shows, when the first video stream is playing alone, as in the previous case, it receives the full 25 fps. However, when the second video streams becomes available, it plays at 25 fps and the video stream from the friend's house is reduced to 5 fps. This is certainly an improvement, however, nothing happens when the colleague gets up and returns since there is still no system support for context.

Finally, consider the scenario with Corvus running on the laptop, as shown in Figure 5.2(c). As in the previous two scenarios, when the first video stream is playing alone, it receives the entire 25 fps. When the colleague's video stream becomes available, it runs at 25 fps and the friend's video stream is reduced to 5 fps as in the priority quality of service scheme. When the colleague gets up from their desk however, the friend's video stream increases to 25 fps and the colleague's slows to 5 fps since Corvus reacts to the context about presence in the video field. Finally, when the colleague returns, Corvus reacts by increasing the colleague's video stream to 25 fps and decreasing the friend's stream to 5 fps. This example shows that the use of context facilitated by Corvus leads to appreciable increases in system usability and performance.

5.4 Contributions and Future Directions

With advances in mobile computing technologies and an increase in pervasive computing devices in the user's environment comes the possibility of leveraging cross-layer resource information to make more effective mobile computing platforms. We have developed a system architecture and a prototype, the Corvus mobile system, that enables resource information to be used by the operating system and adaptive applications.

The implementation of Corvus and its integration into the GRACE system allows the development and testing of adaptive protocols, applications, and mechanisms in all layers of the system.

Future research directions include developing the policy enforcement modules for different parts of the system. Since Corvus is designed to run on a single system, it is important to make sure that all applications running on the system are well-behaved. The same method of policy enforcement could be used to support applications that cannot react by reading from the blackboard.

Chapter 6

Application Energy Conservation

The final piece of this work is the application layer. Application energy conservation techniques aim to reduce energy consumption by altering the behavior of the application, which can affect both resource usage and the quality of the data (*e.g.*, reducing the quality of a video stream reduces the amount of data transmitted). Such application adaptations can be based on both energy conservation goals and changes in the availability of resources. For example, Odyssey [33] and Puppeteer [9] use filtering to drop certain parts of a document depending on the amount of bandwidth or the battery lifetime to reduce the network costs (both in terms of bandwidth and energy) for transmission. Both of these solutions adapt the application data to the current network conditions. Essentially, these approaches change the amount of data transmitted based on the bandwidth that is currently available. Therefore, they do not need to consider the CPU energy necessary for such adaptation since it is performed on a proxy. However, for compression algorithms (*i.e.*, video encoding or lossless data compression), the computation necessary to adapt the data stream can result in a significant amount of energy. Therefore, the CPU costs cannot be ignored. Such compression algorithms can be divided into two categories, lossy and lossless.

In the context of lossy compression algorithms, such as multimedia encoding, there are a number of options for applications that could be built into the system [2, 1, 43]. Fundamentally each type of multimedia encoding allows a tradeoff between amount of data sent and the quality (*e.g.*, in terms of frames per second) and computation cost for the creation of the data. However, these applications have typically been developed to reduce the amount of bandwidth necessary to transmit data to support performance over low-bandwidth links. The energy increases or decreases due to a change in the computation costs have not been analyzed. Therefore, such applications must take into

account both the amount of CPU cycles used and the amount of data created to make adaptation decisions in the face of energy constraints. This is precisely the work this thesis accomplishes.

In the context of lossless compression algorithms, there is a body of research that investigates the relationship between CPU energy for data processing (e.g., compression) and the energy consumption for transmitting the processed data. Barr, *et. al.* [5] analyze the total system energy consumption for various lossless compression algorithms, including energy consumption from the network and CPU. While this work points the way to developing system-wide energy conservation techniques, they do not consider adaptation at any layer (*i.e.*, they do not provide an adaptive solution that responds to dynamic changes in both resource availability and cost). Their research only measures and analyzes the system costs of the compression algorithms, showing where energy is expended in the system for each compression algorithm. Our research focuses on enabling adaptive applications that can react to highly dynamic network conditions. Such dynamic network conditions change the costs associated with the transmission of data and therefore determine the most energy efficient configuration for the application.

Adaptive applications have a range of operating points that may impact the amount of computation and the amount of data generated, which can be directly translated into the amount of bandwidth needed to transmit the data. In the context of energy conservation, it is interesting to look at applications that can tradeoff computation and bandwidth, enabling energy conservation by choosing the most energy-efficient operating point. The most common of such applications use compression algorithms, where more compression usually requires more computation. In general, compressing the data before transmission can save total energy for transmission when the cost of transmitting the uncompressed data is more expensive than the cost of compressing the data and then transmitting it. To determine an energy-efficient operating point (*i.e.*, the most efficient amount of compression), an adaptive application needs information about resource cost and availability, specifically CPU and network resources. This model fits well with our cross-layer adaptation and energy model, where each layer implements an algorithm for energy conservation and exports only the information necessary to adapt to changes in cost and availability.

An energy-aware adaptive application can use the system energy model developed in Section 2 to determine what information from the other layers should be used to adapt its own functionality.

Adaptations performed in other layers, such as the network adaptations discussed in Section 3, change the values that are used for adaptation but do not change the type of information needed.

6.1 Adaptive Data Transfer

To evaluate the use of cross-layer information in the application layer, we developed two applications: an adaptive file transfer application and an adaptive video encoder. The adaptive file transfer program uses the zlib compression library [14] to encode files, which provides 9 levels of compression, each compressing different amounts with varying costs in terms of the number of CPU cycles to accomplish the compression. Essentially, the file to send is broken into a number of frames, each frame is encoded using one of the compression schemes, and then the frame is sent. The adaptive video encoder uses the GRACE video encoder library [43], based on H.263 [42]. This video library contains 15 encoding schemes that vary the amount of computation needed to encode each frame and the amount of data produced by the computation for each frame. Each raw frame of the video to send is encoded using one of the encoding schemes, and then sent. Therefore, to determine the most efficient encoding scheme, an adaptive application needs information about the per-byte network energy costs and the per-cycle CPU energy costs.

The challenge to choosing the correct encoding scheme is that the choice of encoding scheme is driven E_{trans}, which fluctuates with changes in $\frac{D''}{D'}$ and changes in R. To capture these changes, we define the estimated network cost per-byte to send (Ω_{net}) as follows:

$$\Omega_{net} = \overline{\frac{D''}{D'}} \times \frac{P_{card}}{R},$$

where $\overline{\frac{D''}{D'}}$ is the average data expansion for the transport layer and R is driven by network conditions, which fluctuate with changes in noise in the channel. In addition to Ω_{net}, the application also needs the estimated CPU cost per-cycle (Ω_{CPU}) defined as follows:

$$\Omega_{CPU} = \frac{P_{CPU}}{F}.$$

To determine the expected energy consumption for a given encoding scheme ($E^{(m)}$), where m

is the encoding scheme, let D be the size of the unencoded data, $D'^{(m)}$ be the size of the data after encoding with scheme m, and $C^{(m)}$ be the number of cycles needed to encode the data using encoding scheme m. If no encoding is used ($m = 0$), the energy consumed is driven purely by the network costs:

$$E^{(0)} = D \times \Omega_{net}.$$

When encoded data is used, the energy consumption must include the computation cost and the effect on the reduction in the data transmitted:

$$E^{(m)} = (D'^{(m)} \times \Omega_{net}) + (C^{(m)} \times \Omega_{CPU}). \tag{6.1}$$

Therefore, the application should send unencoded data when $E^{(0)} < E^{(m)}$, for all m; otherwise, the application should choose the encoding level that minimizes $E^{(m)}$.

Across the frames in a stream, the performance of any encoding scheme varies, in terms of the encoding efficiency as well as the number of cycles needed to encode the data. Therefore, a prediction of the encoding efficiency and the number of cycles needed for each encoding scheme can be used to determine the most energy-efficient scheme. Since, for many data sets, the encoding efficiency of the stream is fairly smooth. A standard weighted moving average can be used to update the predictions for both encoding efficiency and number of cycles. The update equation for cycle count is:

$$C^{(m)} = (\alpha \times C^{(m)}) + ((1 - \alpha) \times C),$$

where $C^{(m)}$ is the prediction for cycle count for scheme m. A similar approach can be used for $D'^{(m)}$.

Since no information is known about C or D' for any of the schemes before a stream starts, each encoding scheme is probed during the transmission of the first M frames, where there are M encoding schemes available, initializing $C^{(m)}$ and $D'^{(m)}$. The cost of this probing comes from using the less efficient schemes during the the initialization period.

During run-time, Equation 6.1 is used to determine $E^{(m)}$ for $1 \leq m \leq M$ to determine the optimal encoding scheme to be used. Since the encoding efficiency of a given encoding scheme can vary, predictions developed from one data file are not useful for other files, which is the reason for

```
ApplicationAdapt()
 1  MAXLEVEL : number of encoding schemes (M)
 2  data : data frame
 3  start, stop : cycle counts
 4  next : encoding efficiency
 5  data_size ← sizeOf(data)
 6  if initialize
 7     then count(start)
 8          encode(data, next)
 9          count(stop)
10          if next++ > MAXLEVEL
11             then initialize ← false
12     else next ← Cost(noEncode)
13          for i ← 1 to MAXLEVEL
14          do if next > Cost(i)
15                then next ← i
16                     count(start)
17          encode(data, next)
18          count(stop)
19  updateCycleHistory((stop − start), next)
20  updateEffHistory(data_size, sizeOf(data), next)
```

Figure 6.1: Adaptation Algorithm

initializing all values at the beginning of each new stream.

To prevent a particularly bad efficiency or cycle count from eliminating a particular scheme completely, the history of each unused scheme is decayed by a small percentage, essentially making them cheaper and causing them to be probed after a long time of inactivity.

6.2 Adaptive File Transfer Application

To evaluate the effectiveness of our cross layer adaptive design, we use a prototype file transfer application (aFTP) with adaptive lossless compression and the adaptive network algorithm, both implemented on a Linux system running Fedora Core 4. We use this implementation to validate our cross-layer adaptation approach and the use of our system energy model to drive the adaptations. In this section, we first present the channel and CPU model used for our experiments and then present the energy savings achieved by our adaptive application, demonstrating the saving capabilities of a cross-layer adaptive system.

Our prototype is an implementation of our adaptive application on a Linux system. For experimental repeatability, we use traces of network behavior to drive the variations in channel. These

traces are feed into the implementation to drive the network adaptations. Such variations determine the cost of the network and the available bandwidth. These traces are generated in two ways. First, we gather actual traces of a wireless IEEE 802.11b channel using a Linux machine. Second, the channel is modeled using a Rayleigh Flat Fading Model [47], resulting in variations in the noise in the channel on the order of a packet. A two-state Markov shadowing model is used to produce coarser grain noise variations in the channel. The results in this section are produced by averages runs of the system over both types of traces. The traces produce similar results with the standard deviation between runs being very small.

We model the energy consumed for transmission of data as presented in Section 2.1 for a Cisco Aironet 350 Series PC card [7]. The Aironet supports data rates of 1, 2, 5.5 and 11 Mbps, transmit power levels of 5, 20, 30, 50, and 100 mW, and has power consumptions of 2.24W for transmit mode, 1.35W for Receive/Idle mode, and 0.075W for sleep mode. Variations in network energy from TPC span a narrow range. Therefore, while the application takes into account both the transmit power level and the bandpass modulation scheme chosen by the network adaptation, the choice of configuration is driven by the choice of the bandpass modulation scheme, as we demonstrate in our work on Pincher [21]. Real-time energy consumption measurements are not used because current hardware requires a full reset of the network interface to change bandpass modulation schemes or transmit power levels. Also, current hardware consumes the same amount of energy for each transmit power level. This is because TPC is implemented in current interface cards for use in topology control and not energy conservation. For our experiments, we assume costs for switching between bandpass modulation and transmit power level settings are negligible.

We model the CPU energy based on the Intel StrongARM SA-1110 [26]. The SA-1110 can run either at 133MHz and 240mW or 206MHz and 400mW. Since this work does not consider time bounds on the data transmission, the processor is always run at the lowest speed, which yields the most energy-efficient computation.

For the experiments in this section, we run the prototype, tracking the number of cycles used and the number of bytes transmitted. These figures are used to calculate the energy consumption. The network bandwidth is emulated using the network trace files.

Our experiments consist of testing our adaptive file transfer application and our network al-

gorithm on a laptop running Linux. The energy cost for data transmission using the adaptive application includes the total transmission energy (including any retransmissions) plus the energy used to compress the data (if any compression is selected) plus the energy consumed by the adaptation algorithm itself. The application is run multiple times with different trace files and average results are presented.

To test the flexibility of our system, we evaluate the impact of transmitting various types of data using the file transfer program. Some of the data compresses very well (*e.g.*, web pages without images) while other data compresses poorly (*e.g.*, JPEG images). The zlib library provides an interesting adaptation space because the levels of compression are not linearly ordered for any given selection of data (*i.e.*, level 9 is not necessarily more expensive than level 5).

We compare the results achieved by our adaptive application to three other algorithms. The first algorithm simply sends the data uncompressed. All results presented in this section are normalized to this algorithm's results. The next algorithm uses the best single compression level, which is found by sending each file with each single configuration over the same network traces. Once all of the data is gathered, the best configuration is the one with the lowest total energy consumption. The third algorithm is the ideal algorithm, which finds the best configuration at each frame by running each frame using all of the configurations and only counting the one with the lowest energy consumption. The second and third algorithms require information about the future state of the network, as well as information about the future performance of the compression algorithms on the data set. Obviously, these algorithms cannot be implemented on a real system because time-travel is not currently available [12]. Our experiments demonstrate that the adaptive file transfer application reactively chooses compression levels that minimize total system energy consumption, including situations when no compression should be used, without requiring knowledge of the future network state or compression algorithm performance.

To show the frequency with which our adaptive application changes compression levels as a reaction to changing costs in the network layer, we present Figure 6.2. The y-axis depicts the compression level used by the application. Zero is no compression. The x-axis represents the frame number. This trace is from a run sending web page data. The first nine frames run up the compression levels. This represents the beginning of the algorithm initializing the history values

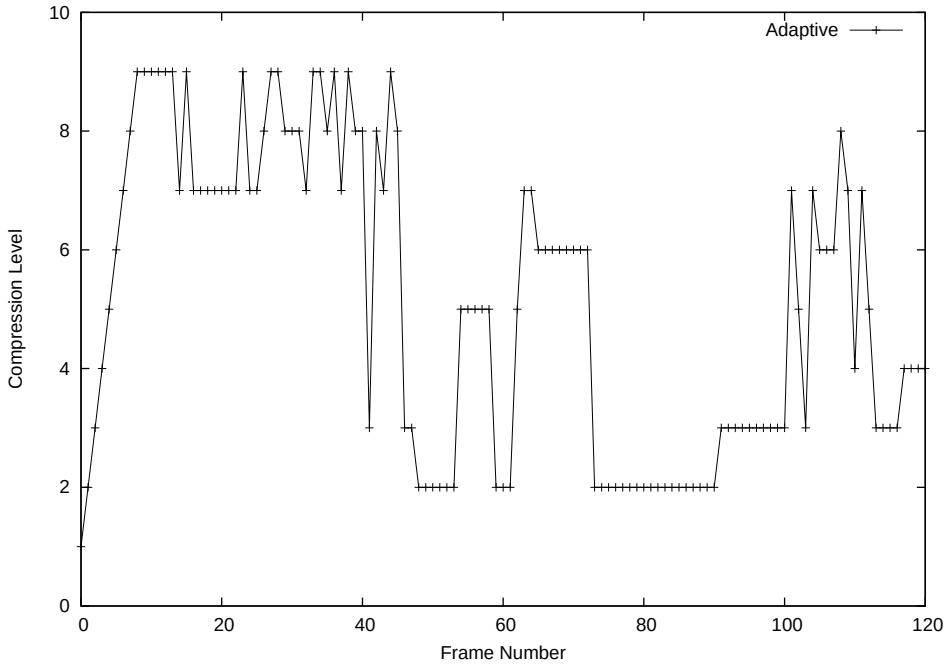

Figure 6.2: Compression Level Trace

for each of the compression levels. The ideal algorithm trace is similarly dynamic. This shows that to produce the most energy efficient results, an adaptive algorithm must react to the dynamics in the network.

The energy consumption of the adaptive file transfer application, the best single compression level, and the ideal algorithm all normalized to sending the file with no compression for four different file types (kernel-code, web, man pages, and JPEG) is depicted in Figure 6.3. The best performance is seen in the kernel code, which contains a lot of white space and is very compressible. Savings of nearly 50% over no compression are seen in this case. The JPEG data that compresses poorly causes the adaptive file transfer application to spend more energy than a non-adaptive application. This is because the adaptation algorithm itself has an energy cost, even though it chooses no compression. Also, every 50 frames, the adaptation algorithm triggers a reinitialization of the history due to the fact that no compression is chosen every time. However, the adaptive application still performs much better than the best single compression level, which performs about 600% worse than the no compress case. The poorer performance of our adaptive application suggests adapting the `resetThreshold`, which is the topic of future work.

Figure 6.4 depicts the energy consumption of the adaptive file transfer application, the best

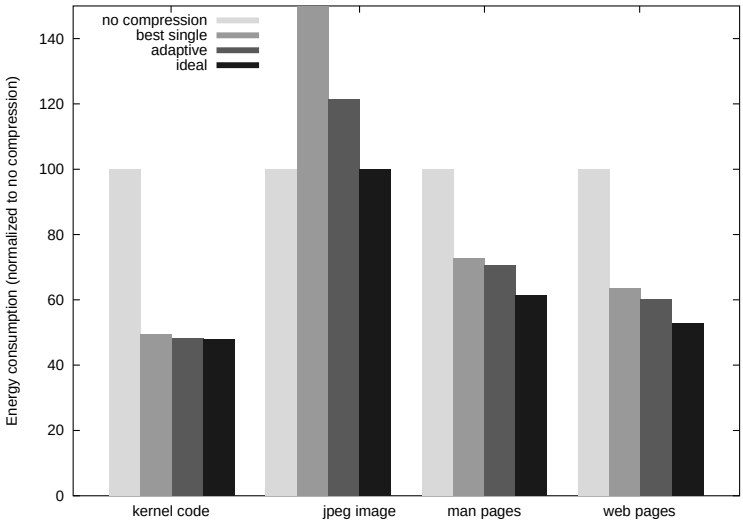

Figure 6.3: Energy Savings: Adaptive File Transfer Application

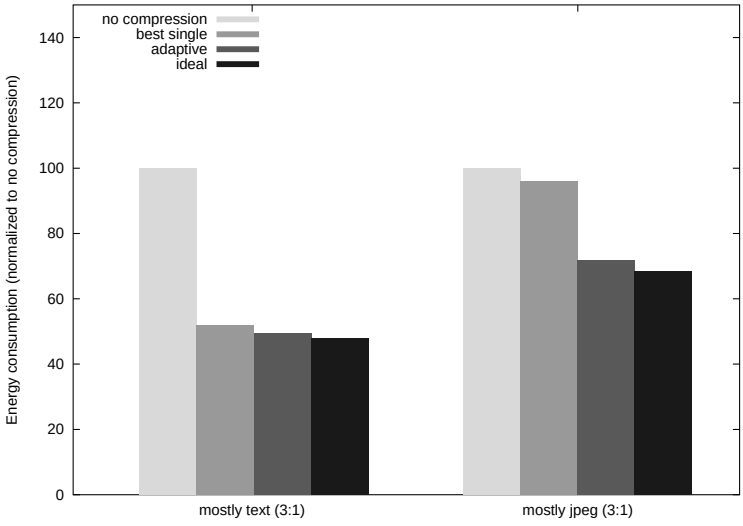

Figure 6.4: Energy Savings: Adaptive File Transfer Application

single compression level, and the ideal algorithm all normalized to sending the file with no compression for two mixed-media files. The difference between the mixed-media files is in the ratios in which HTML data and JPEG data are mixed. As expected, as the amount of JPEG images in the transfer increases, the best single compression level performs more poorly, while the adaptive application remains very close to the ideal algorithm.

These experiments show that significant energy savings can be achieved when cross-layer information is used by an adaptive application to trade off different system resource costs. The adaptive application continuously performed very near to the ideal algorithm and always better than the best single compression algorithm.

6.3 Adaptive Video Encoding Application

The adaptive video encoding application (aVE) is implemented as an application on top of Reaper using the adaptation algorithm presented in Figure 6.1. Our testbed consists of a laptop, an IEEE 802.11b base station, and a wired host. The wired host is the receiver and runs Fedora Core 4 with the user-space implementation of Reaper. The laptop acts as the sender and also runs Fedora Core 4 and Reaper. The laptop has an IEEE 802.11b interface and a Pentium M processor. The processor runs at 600MHz and $P_{CPU} = 3W$. $P_{card} = 980mW$ and the MAC layer dynamically adapts its bit rate, R, according to the channel conditions. For experimental purposes, we set the card to a fixed R to evaluate the long term effects of different CPU and network costs. To determine the energy consumption, the CPU rate and cost are available via the Linux `/sys` file system. For these experiments, the receiver is one wired hop away from the base station. All experiments are run on an active wireless network with uncontrolled cross traffic. Therefore, our experiments represent real-world scenarios where a wireless host must compete for channel time and network bandwidth.

Without cross-layer information about network and CPU costs and bandwidth availability, an adaptive application cannot be optimized to save energy. However, our evaluation show that aVE coupled with Reaper can effectively adapt to the dynamics of network cost and availability. To demonstrate this, we evaluate aVE using a standard suite of video data. The compressibility of the data varies across videos and across frames within a single video. The video files are CIF format and frames are fragmented into one to thirty-five 1440B fragments depending on the encoding scheme

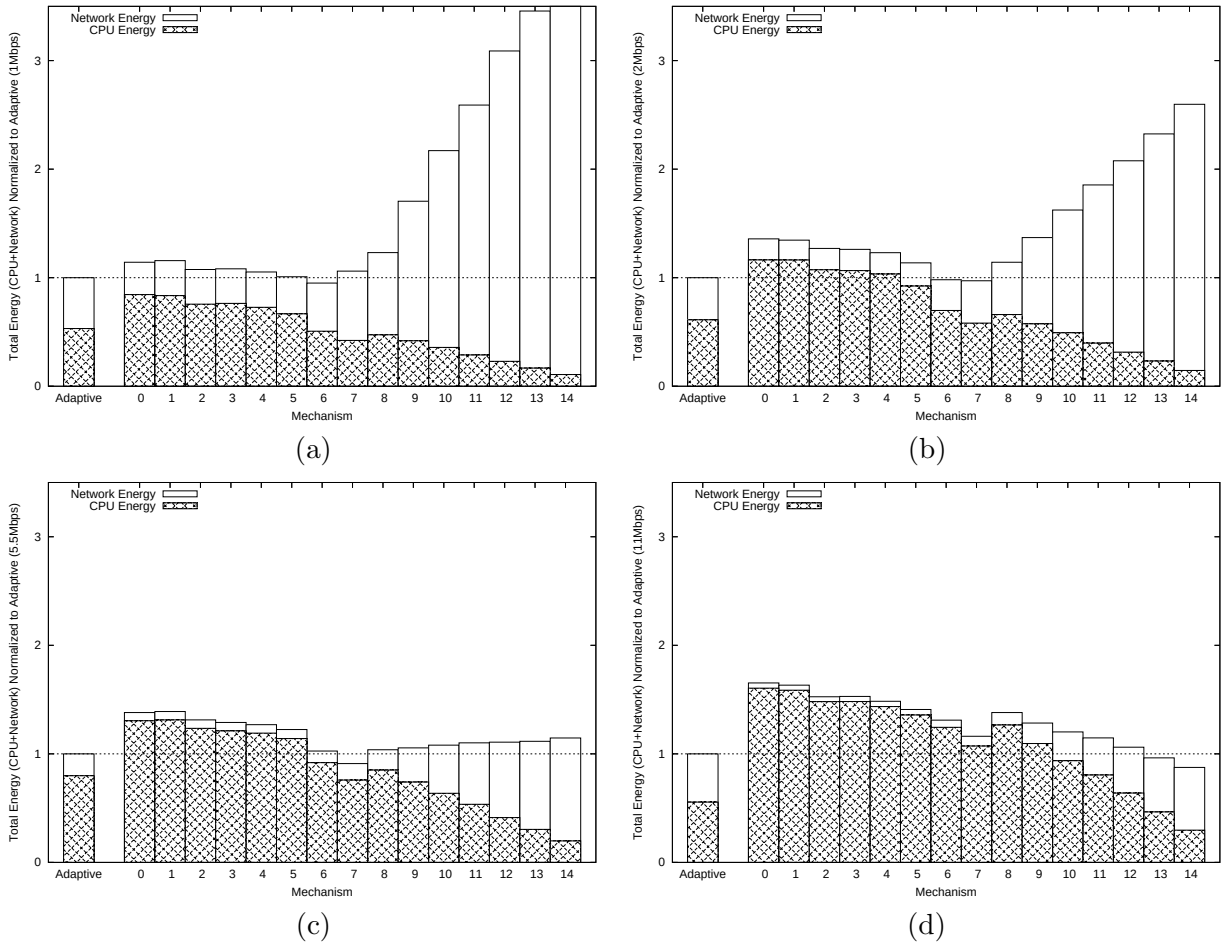

Figure 6.5: Energy Performance at (a) 1 Mbps, (b) 2Mbps, (c) 5.5Mbps, (d) 11Mbps

chosen by the application.

For these experiments, the relative cost of the CPU and the network depends on the current rate at which the network is operating. For low rates, the network is quite expensive compared to the CPU, while for the highest rate the network is cheaper than the CPU. Therefore, no one fixed encoding scheme is likely to be optimal for all network conditions. To show this, we compare each fixed encoding scheme with aVE for each of the network bit rates (1Mbps, 2Mbps, 5.5Mbps, and 11Mbps). We will show that there is no best fixed encoding scheme as the network conditions change, supporting our claim that the application needs cross-layer information to optimize for energy efficiency.

The graphs in Figure 6.3 show the total system energy consumption for each encoding scheme at each bit rate. When the network is running at 1Mbps, encoding schemes 0 through 7 use a

considerable amount of CPU energy and achieve a large amount of data compression. On the other hand, schemes 9 through 14 use very little CPU and achieve little data compression. Scheme 8 employs a technique that uses moderate CPU energy and achieves moderate compression. However, for 1Mbps, scheme 6 uses the minimum system energy. Our adaptive algorithm, aVE, uses less than 5% more energy than this best fixed scheme. This overhead is due to the need to probe the highly inefficient compression schemes. When the network rate is increased to 2Mbps, the best fixed compression scheme is no longer 6 but shifts to 7 because the network is becoming cheaper. Scheme 6, however, still uses slightly less energy than aVE (see Figure 6.3b). However, when the network operates at 5.5Mbps, scheme 6 uses slightly more energy than aVE (see Figure 6.3c). Scheme 7 is still the best fixed encoding scheme. Finally, when the network is operating at 11Mbps, network energy is dominated by CPU energy. Therefore, the encoding schemes that use the smallest amount of CPU use the least amount of energy, as shown in Figure 6.3d. In this case, schemes 6 and 7, which used the least energy at lower network speeds, use 40% and 20% more energy, respectively. Scheme 14, which is the best fixed in this instance, uses as much as 350% more energy than aVE at lower network speeds. In all cases, aVE uses less than 12% more energy than the best fixed scheme. Given encoding schemes or network technologies with greater cost variance, our adaptive system would perform even better compared to the best fixed schemes.

Since there is no best fixed compression scheme, choosing any one fixed compression scheme will result in poor energy efficiency depending on the state of the network. Without cross-layer information to support application scheme adaptation, energy efficiency is impossible to attain, especially in the face of dynamic network characteristics. By using information both about the prevailing network conditions and about current CPU costs, aVE intelligently adapts its encoding schemes and successfully conserves energy in all environments. While some cross-layer information is necessary, the information required is simple, allowing easy development of adaptation algorithms.

6.4 Contributions and Future Directions

We have presented two applications, aFTP and aVE, and used them to show that effective use of cross-layer information results in energy efficient systems. This chapter has demonstrated the ability of the entire adaptive framework developed in the thesis to maintain quality of data as well

as energy efficiency while passing minimal information between the layers of the system.

Future work outside the scope of this thesis includes the development of a system programming API to allow the simple creation of adaptive applications that make use of cross-layer information provided by different layers of the system. Such an API is instrumental in facilitating the actually use of any platform.

Chapter 7

Conclusions And Future Directions

To deal with the mismatch between the rapidly increasing energy demands of mobile multimedia systems and the slow increase in battery capacity, research is being focused on energy efficient mobile system design. Through the use of a system energy model that captures the tradeoffs between adaptations in various system components, we designed a cross-layer adaptive system, including a MAC protocol, a transport protocol, and a suite of applications.

The adaptive MAC layer algorithm, Pincher, achieves energy efficient communication through fast data transmission. Using the mechanisms incorporated in Pincher, we analyze the impact of such adaptations on total system energy consumption. This analysis shows that information about bandwidth must be used by higher layer adaptation algorithms to provide energy efficient adaptations.

The adaptive reliability transport protocol, Reaper, supports adaptive applications with various timing and reliability requirements. This transport protocol exposes information about energy costs, data rate available, and expected packet loss rate for use in application adaptations. Using Reaper, we show that with a non-adaptive application, significant energy savings can still be achieved by make use of information from the application about the data.

Finally, the components of the system architecture are tested using adaptive applications. We have developed an adaptive file transfer application and an adaptive video encoding application. This suite of applications is used to show that greater energy savings can be achieved by using a cross-layer adaptive system than by using discrete adaptations in single layers of the system.

Future research in energy efficient mobile system design can take a number of directions. First, new platforms are available. From small portable devices such as iPaqs that are now capable

of supporting multimedia applications to sensor nodes which are energy constrained even when attempting to send small amounts of data. Second, different network topologies may pose new problems for cross-layer system design, such as ad hoc or mesh networks. Finally, different wireless technologies may fundamentally change the tradeoffs which the network adaptations are creating, such as UWB and GPRS.

Appendix A

Values For Wireless LAN Interfaces

Bandpass Modulation	DBPSK	DQPSK	CCK	CCK
Rate (Mbps)	1	2	5.5	11
SNR Threshold	-2.92	1.59	5.98	6.99

Table A.1: IEEE 802.11b Bandpass Modulation Schemes

Transmit Power (mW)	5	20	30	50	100
Signal Level (dB)	7	13	15	17	20

Table A.2: Cisco Aironet 350 Power Settings

Card Mode	Transmit	Receive/Idle	Sleep
Power (W)	2.24	1.35	0.075

Table A.3: Cisco Aironet 350 Mode Costs

Appendix B

Signal Strength Threshold Calculations

Theoretical, closed form solutions are well known for translating a given SNR into a BER assuming a Rayleigh Flat Fading Model. Proakis [39] gives the theoretical solution for BPSK bit error probability, P^b_{BPSK}, given a fixed attenuation where γ is the instantaneous SNR and $\overline{\gamma}$ is the average received SNR:

$$P^b_{BPSK}(\overline{\gamma}) = Q(\sqrt{2\gamma}) \tag{B.1}$$

In Equation B.1, Q is a standard probability error function, called the *complimentary error function*, given by:

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-y^2/2} dy \tag{B.2}$$

Typically, channels do not, however, have fixed attenuation. Therefore, it is necessary to find a solution that captures the random attenuation expected in wireless channels. Following Hanzo, *et. al.* [20], for random attenuation, the theoretical BER can be obtained by integrating Equation B.1 over the probability density function, $p(\gamma)$.

$$P^b_{BPSK_r}(\overline{\gamma}) = \int_0^\infty P^b_{BPSK}(\gamma) p(\gamma) d\gamma \tag{B.3}$$

To solve Equation B.3, we use the Rayleigh Flat Fading model, with the probability density

function shown in Equation B.4.

$$p(\gamma) = \frac{1}{\overline{\gamma}} e^{-\frac{\gamma}{\overline{\gamma}}} \tag{B.4}$$

The form of the Q function given in Equation B.2 does not yield a solvable form of Equation B.3. However, the Q function is related to erf, another standard probability error function, shown in Equation B.5:

$$Q(x) = 0.5(1 - erf(\frac{x}{\sqrt{2}})) \tag{B.5}$$

Combining Equation B.3 with Equation B.4 and Equation B.5 we obtain [20]:

$$P^b_{BPSK_r}(\overline{\gamma}) = \frac{1}{2\overline{\gamma}} \int_0^\infty e^{-\frac{\gamma}{\overline{\gamma}}} (1 - erf(\sqrt{\gamma}))dy \tag{B.6}$$

To solve Equation B.6, Hanzo *et. al.* [20] note Equation B.6 can be modeled as a standard integral from Gradshteyn's table of integrals [18]:

$$\int_0^\infty e^{bx}(1 - erf(\sqrt{ax}))dx = \frac{1}{b}[\sqrt{\frac{a}{a-b}} - 1] \tag{B.7}$$

By combining Equation B.7 with Equation B.6 and substituting $a = 1$, $b = -\frac{1}{\overline{\gamma}}$, we arrive at a formula for the theoretical BER for BPSK in the presence of random attenuation and Rayleigh Flat Fading:

$$P^b_{BPSK_r}(\overline{\gamma}) = \frac{1}{2}(1 - \sqrt{\frac{\overline{\gamma}}{1+\overline{\gamma}}}) \tag{B.8}$$

Following the same procedure, Hanzo, *et.al.* [20] evaluate BER for QPSK, 16QAM, and 64QAM as follows:

$$P^b_{QPSK_r}(\overline{\gamma}) = \frac{1}{2}(1 - \sqrt{\frac{\overline{\gamma}}{2+\overline{\gamma}}}) \tag{B.9}$$

$$\begin{aligned} P^b_{16QAM_r}(\overline{\gamma}) = \quad & \tfrac{1}{2}[\tfrac{1}{2}(1 - \sqrt{\tfrac{\overline{\gamma}}{10+\overline{\gamma}}}) \\ & +\tfrac{1}{2}(1 - \sqrt{\tfrac{9\overline{\gamma}}{10+9\overline{\gamma}}})] \end{aligned} \tag{B.10}$$

$$\begin{aligned} & P^b_{64QAM_r}(\overline{\gamma}) = \\ & \quad \frac{1}{3}(P^b_{64QAM_1} + P^b_{64QAM_2} + P^b_{64QAM_3}) \end{aligned} \tag{B.11}$$

$$\begin{aligned} & P^b_{64QAM_1} = \\ & \quad \frac{1}{4}[\frac{1}{2}(1 - \sqrt{\frac{\overline{\gamma}}{42+\overline{\gamma}}}) + \frac{1}{2}(1 - \sqrt{\frac{9\overline{\gamma}}{42+9\overline{\gamma}}}) \\ & \quad +\frac{1}{2}(1 - \sqrt{\frac{25\overline{\gamma}}{42+25\overline{\gamma}}}) + \frac{1}{2}(1 - \sqrt{\frac{49\overline{\gamma}}{42+49\overline{\gamma}}})] \end{aligned} \tag{B.12}$$

$$\begin{aligned} & P^b_{64QAM_2} = \\ & \quad \frac{1}{2}[\frac{1}{2}(1 - \sqrt{\frac{\overline{\gamma}}{42+\overline{\gamma}}}) + \frac{1}{2}(1 - \sqrt{\frac{9\overline{\gamma}}{42+9\overline{\gamma}}}) \\ & \quad +\frac{1}{4}(1 - \sqrt{\frac{25\overline{\gamma}}{42+25\overline{\gamma}}}) + \frac{1}{4}(1 - \sqrt{\frac{49\overline{\gamma}}{42+49\overline{\gamma}}})] \end{aligned} \tag{B.13}$$

$$P^b_{64QAM_3} = \tag{B.14}$$
$$\frac{1}{2}[(1-\sqrt{\frac{\overline{\gamma}}{42+\overline{\gamma}}})+\frac{3}{4}(1-\sqrt{\frac{9\overline{\gamma}}{42+9\overline{\gamma}}})$$
$$-\frac{3}{4}(1-\sqrt{\frac{25\overline{\gamma}}{42+25\overline{\gamma}}})-\frac{1}{2}(1-\sqrt{\frac{49\overline{\gamma}}{42+49\overline{\gamma}}})$$
$$+\frac{1}{2}(1-\sqrt{\frac{81\overline{\gamma}}{42+81\overline{\gamma}}})+\frac{1}{4}(1-\sqrt{\frac{121\overline{\gamma}}{42+121\overline{\gamma}}})$$
$$-\frac{1}{4}(1-\sqrt{\frac{169\overline{\gamma}}{42+169\overline{\gamma}}})]$$

Appendix C

MAC Layer Effects On System Energy Consumption

The role of network layer adaptations is to minimize the energy consumed to transmit data. The design of effective higher-layer adaptation algorithms requires an understanding of the range of potential energy savings and the relative impact on total system energy for each optimization mechanism. Essentially, each mechanism has a range of operating points, each of which has a defined power requirement, as well as an impact on the transmission time, which together define the total energy consumption. While each mechanism may have a large range of adaptations and so have a large effect on the targeted component, the impact on the energy consumed by the whole system may or may not be significant. In the context of network adaptations, it is therefore necessary to consider the relative impact of each of the mechanisms discussed in the previous subsections. In this section, we model the impact of TPC, BPM, and power management. This analysis shows which mechanisms cause changes in energy consumption that might impact decisions by higher-layer adaptation algorithms and therefore shows which mechanisms must pass information up to higher layers.

Given differing characteristics of various mobile devices and wireless network devices, the relative impact of each network layer energy conservation mechanism can be captured in the context of a real system by looking at the range of operating points for each network adaptation mechanism. However, network protocols are constrained by the state of the channel and must react to the changing channel conditions. Therefore, these ranges do not represent actual energy savings, but illustrate the impact of each mechanism on total system energy depending on the relative

contribution of the component to total system energy.

The evaluations in this section capture the range of energy savings each technique can achieve over the range of channel conditions as a percentage of total system energy consumption for the creation and transmission of data. In any lossless data compression algorithm, each packet must be compressed, processed through the protocol stack and transmitted, the total energy for which is $E_{CPU} + E_{data-send}$. To cover a range of different types of wireless network cards, we define three models for $E_{data-send}$.

- Model 1, $E_{data-send}$ only includes the P_t component (*i.e.*, $P_{base} = P_{xmit} = 0$), which reflects a device similar to mobile phones where transmission costs out-weight any costs from driving the transmitter or the card.

- Model 2, $E_{data-send}$ includes both the P_t and the P_{xmit} components (*i.e.*, $P_{base} = 0$), which reflects a device that has minimal costs to keep the card on.

- Model 3, $E_{data-send}$ includes all components, which reflects current wireless LAN cards that have high idle costs.

Since $P_{base} = 0$ for the first two models (*i.e.*, there is no idle-time energy consumption), there is no benefit from power management. However, the third model assumes a power management scheme that places the wireless interface into a low power sleep mode, allowing the conservation of P_{base} during idle periods, demonstrating power management's potential energy savings. To model the energy consumption of a modern WLAN device with TPC capabilities, we use a Cisco Aironet 350 series PC Card [7], with available bandpass modulation schemes depicted in Table A.1 and available transmit power levels depicted in Table A.2. The energy consumption of the Cisco Aironet in the different card modes is shown in Table A.3. For this card, the transmission energy costs are broken down as follows: $P_t = [5 - 100]mW$, $P_{xmit} = 980mW$ and $P_{base} = 1275mW$.

For the data compression, each packet of data is read from a file and compressed using the zlib compression utility, using an average of 40,000,000 cycles per packet. E_{proc} captures the CPU processing in the protocol stack, which we measured to be an average of 30,000 cycles for UDP, IP and software MAC processing in Linux. To model current mobile multimedia devices, we set E_{CPU} to reflect a 240mW, 133MHz StrongARM processor [26]. While the exact value of E_{CPU} effects

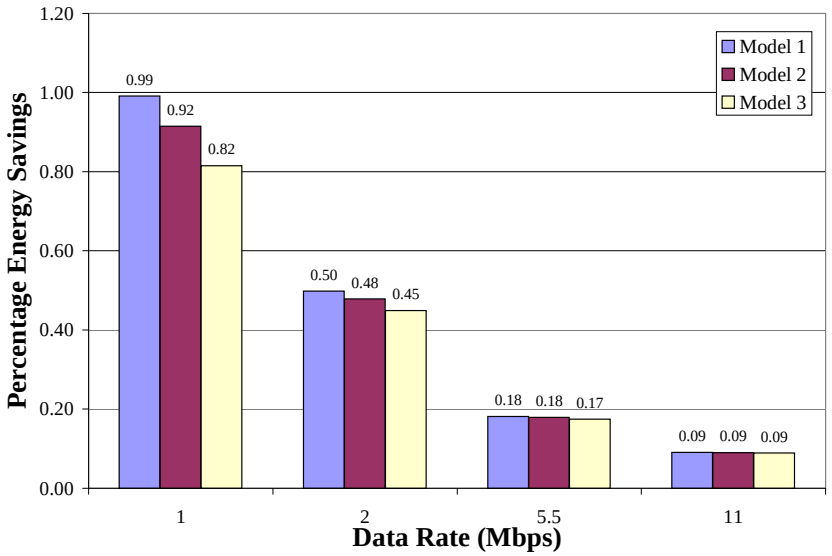

Figure C.1: The Range of Potential Energy Savings using TPC

total energy consumption, variations in E_{CPU} only change the relative contribution of the CPU to total energy consumption and do not impact the absolute effect of network-layer adaptations. We have evaluated the impact of different values of the components of E_{CPU} on the potential savings from internal network adaptations. All cases show the same trends, which we discuss below, with differences only in the magnitude related to E_{CPU}, not the relationship between the internal network adaptations. Therefore, for evaluation of internal network adaptations, E_{CPU} is held constant.

The use of TPC alone only affects the P_t component of $E_{data-send}$. Given a range of P_t values of [5,100], and the other costs as described above, we evaluated the potential energy saving by using TPC for each transmission rate. The range of potential savings is calculated by evaluating E_{app} at the maximum $P_t = 100mW$ and the minimum $P_t = 5mW$. Figure C.1 presents the potential relative savings for each energy model at each rate. TPC has the most impact on Model 1, since P_t dominates $E_{data-send}$. However, as P_{xmit} and P_{base} are added back into $E_{data-send}$, TPC has less impact due to the fixed costs of P_{xmit} and P_{base}. Additionally, TPC has the most impact at slower rates. Since the wireless card is transmitting for a longer time, $E_{data-send}$ increases while E_{CPU} remains fixed. As $E_{data-send}$ becomes a larger component of E_{app}, any potential savings in $E_{data-send}$ have a higher impact on E_{app}. Therefore, TPC has higher potential energy savings at lower data rates and for devices where P_t dominates network costs.

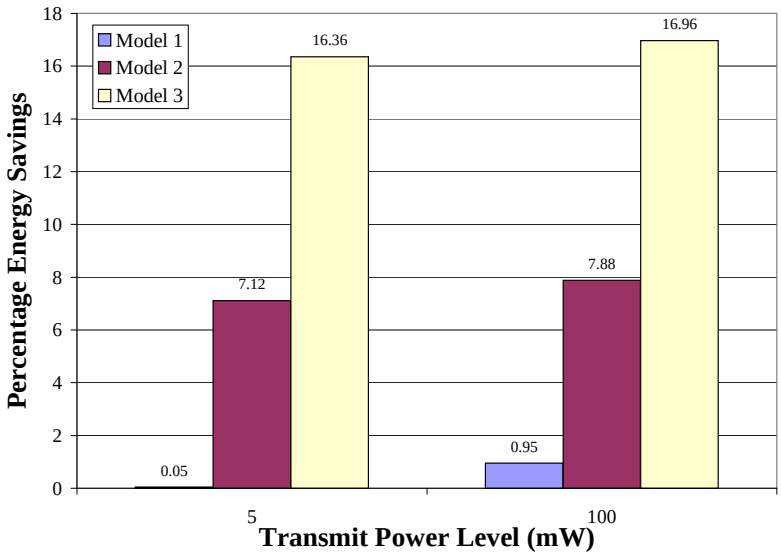

Figure C.2: The Range of Potential Energy Savings using BPM

The relative effect of BPM can be evaluated in a similar manner. Alone, BPM only affects the R component of $E_{data-send}$, affecting the amount of time the network card consumes energy for transmission. The range of potential savings is calculated by evaluating E_{app} at each rate for the maximum and minimum P_t values (see Figure C.2). Since there is no P_{base} or P_{xmit} in Model 1, the impact of BPM is limited. Essentially, the $E_{data-send}$ component of E_{app} is quite small compared to E_{CPU}. However, as P_{base} and P_{xmit} are added back in, $E_{data-send}$ increases significantly when the card is transmitting at slower rates, resulting in a large range of potential savings. Therefore, as the base costs of the network interface grow, transmitting at faster rates saves more energy.

A complete network adaptation layer can use TPC and BPM simultaneously to maximize energy savings. The range of savings is evaluated between the slowest rate at the highest transmit power level and the highest transmission rate at the lowest transmit power level. Again the widest range of savings is seen for Model 3, since it has the largest $E_{data-send}$ (see Figure C.3).

Through these evaluations, we have mapped the space for potential savings from internal network optimizations and so can determine which optimizations should be used in a given environment. For example, a wireless LAN card (*i.e.*, Model 3) has the most potential savings from BPM. However, TPC should also be used to reduce energy consumption as much as possible. However, for a cellular phone (*i.e.*, Model 1), the most potential energy savings derive from TPC.

Therefore, changes in bandwidth will drive higher layer adaptation. Our network layer must

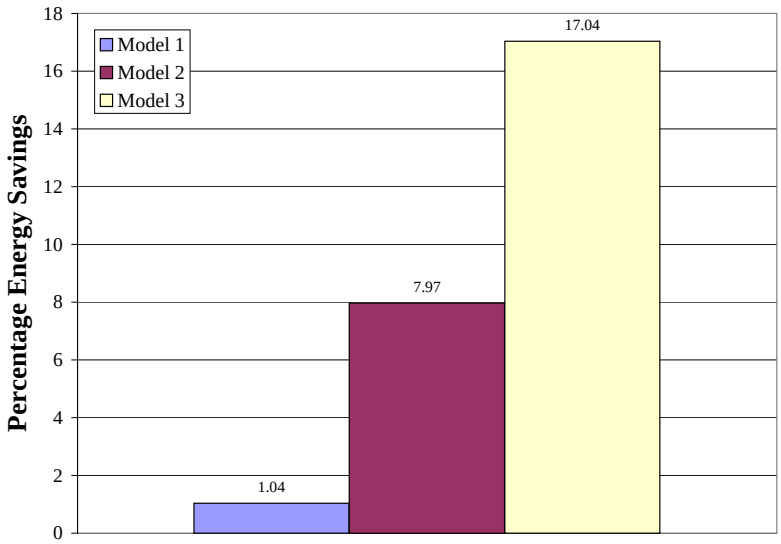

Figure C.3: The Range of Potential Energy Savings using Combined TPC/BPM

therefore have some mechanism to pass information about the current achievable bandwidth to higher layer adaptation algorithms.

References

[1] Mpeg video compression. "http"//www.mpeg.org/".

[2] Speex audio codec. "http://www.speex.org/".

[3] S. V. Adve, A.F. Harris, C.J. Hughes, D.L. Jones, R.H. Kravets, K. Nahrstedt, D. G. Sachs, R. Sasanka, J. Srinivasan, and W. Yuan. The illinois grace project: Global resource adaptation through cooperation. In *SHAMAN*, 2002.

[4] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Ghosts in the machine: Interfaces for better power management. In *Proc. MobiSys 2004*, 2004.

[5] Kenneth Barr and Krste Asanovic. Energy aware lossless data compression. In *Proc. of MobiSys 2003*, 2003.

[6] A. Chockalingam and M Zorzi. Wireless tcp performance with link layer fec/arq. In *Proc. IEEE ICC*, 1999.

[7] Cisco. Cisco Aironet 350 client data sheet. http://www.cisco.com/.

[8] M. Corner, B. Noble, and K. M. Wasserman. Fugue: Time scales of adaptation in mobile video. In *Proc. of the SPIE Multimedia Computing and Networking Conference*, 2001.

[9] Eyal de Lara, D. S. Wallach, and W. Zwaenepoel. Puppeteer: Component-based adaptation for mobile computing. In *3rd USENIX Symposium on Internet Technologies and Systems*, 2001.

[10] L. Donckers, P. Havinga, G. Smit, and L. Smit. Enhancing energy efficient tcp by partial reliability. In *13th IEEE PIMRC*, 2002.

[11] Jean-Pierre Ebert and Adam Wolisz. Combined tuning of RF power and medium access control for WLANs. *Mobile Networks & Applications*, 5(6):417–426, Sept. 2001.

[12] Albert Einstein. *The Collected Papers of Albert Einstein*, volume 2. Princeton, 1989.

[13] L. M. Feeney. An energy consumption model for performance analysis of routing protocols for mobile ad hoc networks. *Mobile Networks and Applications*, 6(3):239–249, June 2001.

[14] Jean-Loup Gailly. zlib. http://www.info-zip.org/pub/infozip/zlib/.

[15] J. Gass, M. Pursley, H. Russell, R. Saulitis, C Wilkins, and J. Wysocarski. Adaptive transmission protocols for frequency-hop radio networks. In *Proc. 1998 IEEE Military Communications Conference*, volume 2, October 1998.

[16] J. Gomez, A. T. Campbell, M. Naghshineh, and C. Bisdikian. PARO: Supporting dynamic power controlled routing in wireless ad hoc networks. *Wireless Networks*, 9(5):443–460, September 2003.

[17] Javier Gomez, Andrew Campbell, Mahmoud Naghshineh, and Chatschik Bisdikian. Conserving transmission power in wireless ad hoc networks. In *Proc. of 9th IEEE International Conference on Network Protocols (ICNP '01)*, 2001.

[18] I.S. Gradshteyn and I. M. Ryzhik. *Table of Integrals, Series and Products*. Academic Press, 1980.

[19] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Drpm: Dynamic speed control for power management in server class disks. In *Proc. of the 30th Annual International Symposium on Computer Architecture*, 2003.

[20] L. Hanzo, Wong C.H., and M.S. Yee. *Adaptive Wireless Transceivers*. Wiley, 2002.

[21] Albert F. Harris III and Robin Kravets. Pincher: A power-saving wireless communication protocol. Technical Report UIUCDCS-R-2003-2368, University of Illinois at Urbana-Champaign, 2003.

[22] Gavin Holland, Nitin Vaidya, and Paramvir Bahl. A rate-adaptive mac protocol for multi-hop wireless networks. In *Proc. 7th ACM International Conference on Mobile Computing and Networking (MobiCom '01)*, 2001.

[23] J.I. Hong and J.A. Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction (HCI) Journal*, 16, 2001.

[24] C. J. Hughes, P. Kaul, S.V. Adve, R. Jain, C. Park, and J. Srinivasan. Variability in the execution of multimedia applications and implications for architecture. In *Proc. 28th Intl. Symp. on Computer Architecture*, 2001.

[25] IEEE 802 LAN/MAN Standards Committee. Wireless LAN medium access control MAC and physical layer (PHY) specifications: Spectrum and transmit power management extensions in the 5ghz band in europe. Draft Supplement to IEEE Standard 802.11 1999 Edition, 2002.

[26] Intel. Intel strongarm* sa-1110 microprocessor brief datasheet. http://developer.intel.com/design/strong/datashts/278241.htm.

[27] S. Iyer, L. Luo, R Mayo, and P. Ranganathan. Energy-adaptive display system designs for future mobile envirnments. In *Proc. of 1st International Conference on Mobile Systems, Applications, and Services*, 2003.

[28] A. Kamerman and L. Monteban. WaveLAN-II: A high-performance wireless LAN for the unlicensed band. *Bell Labs Technical Journal*, 1997.

[29] Ramana Kompella and Alex Snoeren. Practical lazy scheduling in wireless sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems*, 2003.

[30] R. Kravets, K. Calvert, P. Krishnan, and K. Schwan. Adaptive variation of reliability. In *HPN 1997*, 1997.

[31] Robin Kravets and P. Krishnan. Power management techniques for mobile communication. In *Proc. Fourth ACM International Conference on Mobile Computing and Networking (MOBICOM'98)*. 1998.

[32] Jia-Ru Li, Sungwon Ha, and Vaduvur Bharghavan. Hpf: A transport protocol for supporting heterogeneous packet flows in the internet. In *IEEE INFOCOM*, 1999.

[33] Brain Noble. System support for mobile, adaptive applications. *IEEE Personal Communications*, 2000.

[34] ns2 Network Simulator. http://www.isi.edu/nsnam/ns/.

[35] C. Papadopoulos and G. Parulkar. Retransmmission-based error control for continuous media applications. In *NOSSDAV*, 1996.

[36] Sassan Pejhan, Mischa Schwartz, and Dimitris Anastassiou. Error control using retransmission schemes in multicast transport protocols for real-time media. *IEEE/ACM Transactions on Networking*, 1996.

[37] Christian Poellabauer and Karsten Schwan. Energy-aware traffic shaping for wireless real-time applications. In *Proc. RTAS 2004*, 2004.

[38] Balaji Prabhakar, Elif Uysal-Biyikoglu, and Abbas El Gamal. Energy-efficient transmission over a wireless link via lazy packet scheduling. In *Infocom*, 2001.

[39] J. G. Proakis. *Digital Communications*. McGraw Hill International Editions, 3rd edition, 1995.

[40] Daji Qiao, Sunghyun Choi, Amit Jain, and Kang G. Shin. Miser: An optimal low-energy transmission strategy for ieee 802.11a/h. In *Mobicom*, 2003.

[41] R. Ramanathan and M. Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. *Mobile Networks & Applications*, 3(1):101–119, June 1998.

[42] Telenor Research. Tmn (h.263) encoder/decoder, version 2.0. http://www.xs4all.nl/ roalt/h263.html.

[43] D.G. Sachs, S. Adve, and D.L. Jones. Cross-layer adaptive video coding to reduce energy on general-purpose processors. In *Proc. on the International Conference on Image Processing (ICIP'03)*, 2003.

[44] J. Seo, N. Sung, S. Lee, N. Park, H. Lee, and C. Cho. An adaptive type-i hybrid-arq scheme in a tdma system over a non-stationary channel. In *First Workshop on Resource Allocation in Wireless Networks*, 2005.

[45] Tajana Simunic, Luca Benini, Peter Glynn, and Giovanni De Micheli. Dynamic power management for portable systems. In *Proc. 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, 2000.

[46] R. Sinha and C. Papadopoulos. An adaptive multiple retransmission technique. In *NOSSDAV*, 2004.

[47] Bernard Sklar. *Digital Communications: Fundamentals and Applications*. Prentice Hall PTR, 2nd edition, 2001.

[48] IEEE Computer Society. Tcp rfc 2581, April 1999.

[49] IEEE Computer Society. Rtp rfc 3550, July 2003.

[50] William Stallings. *Data and Computer Communications*. Prentice Hall, 2004.

[51] J. Staudinger. Issues and trends in mobile cellular transmitter power amplification. In *Wireless Circuits, Interconnection, and Assembly Workshop*, 1996.

[52] Mark Stemm, Paul Gauthier, Daishi Harada, and Randy Katz. Reducing power consumption of network interfaces in hand-held devices. In *Proc. 3rd. International Workshop on Mobile Multimedia Communications*, 1996.

[53] B. Wah, D. Lin, and X. Su. A survey of error-concealment schemes for real-time audio and video transmission over the internet. In *Proc Int Symposium on Multimedia Software Engineering*, 2000.

[54] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *Proc. of 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[55] Michele Zorzi and Ramesh R. Rao. Error control and energy consumption in communications for nomadic computing. *IEEE Transactions on Computers*, 46(3):279–289, 1997.

Author's Biography

Albert F. Harris, III was born on December 21, 1973. He received a B.A. in Philosophy and a B.S. in Computer Science from Rockhurst University in 2000. He received his Master's Degree in Computer Science from the University of Illinois at Urbana-Champaign in 2002 and went on to complete his Ph.D. there in 2006. He was part of the Modus wireless networking research group under his advisor Robin Kravets. His research interest include energy efficient wireless cross-layer wireless system design. He also has interest in sensor network protocol design.