Network Support For Energy Efficient Wireless Communications

by

Haijin Yan

(Under the direction of David K Lowenthal)

Abstract

In mobile devices, the wireless network interface card (WNIC) consumes a significant portion of overall system energy. One way to reduce energy consumed by a device is to transition its WNIC to a lower-power *sleep* mode when data is not being received or transmitted.

This thesis investigates network layer support for for energy efficient wireless communications. We propose a set of *client-centered* techniques to conserve energy consumption for well-known mobile applications, namely web browsing and TCP downloads. The basic idea behind our network layer energy-saving techniques is that the client actively tracks connections, shapes traffic if necessary, predicts packet arrival time, keeping the WNIC in high-power mode only when necessary.

For web browsing with concurrent connections, our technique tracks the connections to identify intervals in TCP streams where putting the WNIC in *sleep* mode during those intervals is safe and profitable. When upper applications are large file TCP downloads, our technique increases the amount of time that can be spent in *sleep* mode by shaping the traffic.

Our techniques are compatible with standard TCP and do not rely on any assistance from the server, a proxy, or IEEE 802.11b power-saving mode. We demonstrate the effectiveness of our techniques by running comprehensive experiments under both emulated environments and the real Internet. We compare our results with regular TCP, PSM, and BSD whenever possible.

Results show that our technique combines the performance of regular TCP with nearly all the energy-saving of PSM during web browsing, and we save more energy than PSM during client think times. Over an entire web browsing session (downloads and think times), our scheme saves up to

21% energy compared to PSM and incurs less than a 1% increase in transmission time compared to regular TCP. In the case of large file downloads, results show that compared to baseline TCP, our scheme saves over 50% energy in the best case with a transmission time increase under 8% and on average saves 27% energy with a transmission increase of 20%.

INDEX WORDS: Energy, Network, Wireless, TCP, 802.11, Web, FTP

Network Support For Energy Efficient Wireless Communications

by

Haijin Yan

B.A., The University of Electronic Science and Technology of China, 1995

M.S., The University of Electronic Science and Technology of China, 1998

Ph.D., The University of Georgia, 2005

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2005

Network Support For Energy Efficient Wireless Communications

by

Haijin Yan

Approved:

Major Professor: David K Lowenthal

Committee: Suchendra Bhandarkar
Kang Li
K. Roscoe Davis

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2005

Dedication

To my wife, Meng Deng, and my parents.

Acknowledgments

Obtaining a Ph.D is viewed as a significant achievement for many Chinese. I am overwhelmed when I recall all the people who have helped me to get this far during the past years.

First and foremost, I would like to thank my advisor, David Lowenthal, for his constant support, guidance, and inspiration. He's been a great advisor, giving me invaluable advice in both research and life, as well as preparing for my future career. I have greatly benefited from his spirit and attitude.

I also want to thank Dr. Kang Li for his input and help for my dissertation. Dr. Kang Li brought insightful comments and constructive criticisms. He also contributed the active probing idea and other very useful techniques for doing experiments and writing papers in network area.

I am also grateful to Professor Suchendar Bhandarkar and K. Roscoe Davis for serving on my committee and providing valuable feedback on the idea presented in this thesis. Those suggestions are extremely useful to me. I want to thank former faculty member Scott Watterson for his collaboration and contribution in all of the projects.

I have a very productive collaborations with Rupa Chrishnan, Ed Cashin, and Barry Rountree, who have contributed significantly to this thesis.

I would like to thank all my teachers, from elementary school through graduate school, for providing me with a great education.

Most important of all, I would like to express my gratitude to my family for being an unstinting source of support and encouragement. I do not know how would I survive during those paper rejection times if without their supports. My wife, Meng Deng, should take the most credit from this work. With her support, we walk through the hard time and this has become the most precious memory in my life. There are no words can express my thankful to her! Last but not the least, my little brother, Yujin Yan, have been a great brother and have given me excellent life philosophies and career advices.

This dissertation study was supported by a State of Georgia Yamacraw Grant as well as NSF grant CCR-0234285.

Table of Contents

List of Figures

Chapter 1

Introduction

During the past decade, both computing technology and wireless communication technology have advanced greatly. The rapid development and deployment of wireless technology changes not only the way people communicate with each other, but also the way businesses work and information is shared.

As stated in the Business Scenario [39]: "Mobile Computing and Communications are increasingly used in a wide range in government, commercial, manufacturing, service, and other organizations. They help to minimize time and maximize availability and competitive advantage."

The prosperous growth of wireless technology has enhanced human productivity and brought convenience and value to all aspects of human lives. For example, the deployment of 2.5G and 3G networks virtually provides connectivity for everybody on the planet. The proliferation of 802.11 wireless LANs breaks the last mile bottleneck and provides "always online" Internet accesses. The desire for information at your fingertips has greatly and will continue to power the growth of mobile and wireless technology.

However, new challenges come with the rapid development of wireless technology. Network security is a major problem. The inherent insecurity of wireless communications and mobile devices increases the importance of secure wireless networks. As more and more people rely on battery powered mobile devices, energy dissipation becomes another critical issue that impedes the development of mobile computing. Among other issues, QoS guarantees and seamless mobility support are also important factors that affect the design, implementation and utilization of wireless technology.

For the past years, there has been significant research effort invested into wireless communication technologies across industry, academia, and standard groups. Although there has been some significant technology advances, it is clear that wireless technology is still in its infancy, and more effort is needed to better understand wireless networks in every aspect. We observe that energy conservation is one of the active areas that attracts researchers, as it is critical to the sustainable development of wireless technology.

This dissertation puts forth the thesis that *Network Support for Energy Efficient Wireless Communication*, the adaptation of network protocols for energy conservation, is an essential part of a comprehensive energy management solution. Energy management is a broad idea in system research. At the hardware level, energy management solely means low-power circuit design. At the higher level of the system, it is viewed as a resource management problem by treating the limited energy supply as an important resource. Hardware level energy conservation techniques have been proven effective. However, when considering the system as a whole, higher level involvement in energy management is essential. The current research trend in power management has demonstrated that system level power management is more promising than simply attacking the problem at the hardware level.

This dissertation focuses on adapting high level network protocols for energy efficient wireless communications. Simply stated, energy efficient wireless communication is a mechanism that reduces energy consumed by a wireless network interface card(WNIC) by transitioning it to a lower-power *sleep* mode when data is not being received or transmitted. We propose a set of client-centered techniques for saving energy on WNIC for representative applications. We concentrate our efforts on handling TCP streams initiated by the client to a server via request/response. Our client-side implementation can be easily deployed at individual end hosts without any modifications.

The rest of the chapter is organized as follows. We begin in Section 1.1 with a discussion of the rapid growth of mobile computing and applications. We also give a brief statistical analysis on client network usage characteristics in Section 1.2. In Section 1.3, we discuss the importance

of power-aware computing in many aspects and identify specific challenges for energy efficient communications. We then give our research philosophy and design principles in Section 1.4. We conclude in Section 1.5. with an outline of the rest of this thesis.

1.1 Wireless Technology

The development of wireless technology increases productivity. As a representative wireless technology, IEEE 802.11-based wireless networks, are widely deployed in a variety of public places such as libraries, airports, and even in cafes and coffee houses like Starbucks. IEEE 802.11 wireless technology was originally developed as a replacement of Ethernet by providing up to 2 Mbps transmission rates. The state-of-the-art 802.11 now supports up to 54 Mbps rates at much lower cost. Other wireless technologies that share the same fast-growing experience on implementation and management are GSM, 3G, WiMax, and WWAN technologies [21]. All those technologies are aim to providing always on-line connectivity for people with mobile devices.

eMarketer reported that the Yankee Group forecasts 72,480 hotspots in the U.S. by 2007, doubling 2005's estimate. Allied Business Intelligence (ABI) says in its report on Wi-Fi that global hotspots will grow from their current level of 28,000 locations to more than 200,000 within five years [38].

Today, people use mobile devices primarily for voice communication, Web access, and personal data management. In the foreseeable future, people will enjoy a variety of mobile applications from their mobile devices. For example, this includes the abilities to publish/subscribe events and messages and to participate in wireless online games and entertainments. The demand for better services has driven the progress of wireless technology in every aspect.

1.2 Wireless Traffic

As wireless network usage patterns evolve with the technology development, it is critical for researchers to understand mobile client network usage patterns. Here, we present results from some prior works that conducted large scale and comprehensive studies on mobile client usage characteristics in wireless networks.

Work in [58] presents results from the trace of network activity in a large production wireless LAN. Over an eleven week period, they traced the activity of nearly two thousand users drawn from a general campus population. They reached the following conclusion after a careful analysis of the trace.

- Traffic: A median host contributes about 350MB traffic over 11 weeks while the busiest host traffered 117GB. 5% hosts cause more than half of the traffic.
- User mobility: Not many hosts move around much. This may due to the lack of seamless handoff.
- Card activity: As expected, short sessions dominate with an around 17 minutes median session length. About 27% of sessions last less than a minute.
- Protocols: 802.11 broadcast MAC frames account for about 2.6% of all the frames. 99.7% packets are IP packets. Among them, 95% are TCP packets and the other 5% are UDP packets. In the application layer, HTTP traffic dominates with a 53% weight. Other traffic like ftp download and peer-to-peer sharing accounts for 19.2%. AOL instant messaging and email service account for 1.5% and 1.3%, respectively.

From the above analysis, we see that although web protocols still were the single largest component of traffic volume, network backup and peer-to-peer file sharing contributed an unexpectedly large amount to the traffic. Also, instant messaging and other online interactive applications are beginning to gain more popularities. A continuous tracking is needed to reveal changing trends.

1.3 Energy Management

Energy supply, which limited by battery lifetime, is a critical factor that influences the functionality and availability of mobile devices. In the post-PC era, reducing energy consumptions in mobile devices in order to extend battery lifetime is one of the major challenges for researchers.

It is not surprising that the energy problem can not be solved by simply increasing battery size and capacity. First, battery technology has improved very slowly over time [71]. Second, it is not feasible to equip mobile devices with heavy batteries. Finally, hardware will always be power-hungry, driven by upper layer applications.

Advances in low-power circuit design have led to the development of energy-efficient hardware components. For example, the Transmeta Crusoe processor [93] and RAMBUS memory chip [64] are hardwares that designed to be able to reduce energy consumptions in mobile devices. The principle is that every joule is precious and thus must be used to perform useful work. Ideally, with energy-efficient components, hardware should expend energy only when they are being used. When idle, they should enter power-saving states to lower power dissipation.

Unfortunately, it has proven to be insufficient to rely solely on the advances in low-power circuit design and hardware power management to meet the growing energy demands of mobile devices. Recently, the community has begun to realize the trend that higher levels of the system must play a more active role in overall energy management. For an integrated solution, lower-level hardware accept directions from higher level applications and systems to capture any opportunities to conserve energy. In principle, system-level approaches seek to minimize the energy usage of every component from all levels of a mobile system.

From a power management perspective, high level involvement in power management promotes the concept of operating systems directed power management (OSPM). OSPM has many advantages. First, energy is a manageable resource. In that sense, it is operating system's duty to schedule and manage energy wisely. Second, hardware energy conservation strategies can yield

better energy savings when directed by the operating system. Third, energy management with high level involvement can support QoS requirements and tradeoffs in a more efficient and flexible way.

1.3.1 ACPI

The Advanced Configuration and Power Interface (ACPI) specification was developed to establish industry common interfaces enabling robust operating system (OS)-directed motherboard device configuration and power management of both devices and entire systems. ACPI is the key element for implementation OSPM on mobile devices.. The latest ACPI version is 2.0 [104].

The basic idea of ACPI is to integrate power management of each component into a well-defined interface specification. For example, ACPI 2.0 interface includes the existing collection of power management BIOS code, motherboard configuration interfaces, Advanced Power Management (APM) application programming interfaces (APIs), and other related specifications.

ACPI evolves the existing interfaces and provides unified interfaces and OSPM concepts that are suitable to all classes of computers including desktop, mobile devices, workstation, and even server machines. The advanced OSPM interfaces provided by ACPI support power management in a more robust and potentially more efficient manner.

Through the ACPI-defined interfaces, hardware and software vendors are strongly encouraged to build OSPM compatible implementations. With the wide adoption of the ACPI standard, power management is becoming a standard module similar to memory management, scheduling, etc. in the design of operating systems.

In short, today's power management rationale is to move power management into the operating system and to use an abstract interface (ACPI) between the operating system and the hardware to achieve the principal goals set forth above.

1.3.2 Wireless Network Interface Power Management

Because wireless network access is a fundamental function for mobile devices, wireless network interface card (WNIC) has become a significant source of consumed energy in mobile devices; in fact, it can in some cases be the single largest power drain in a mobile device. For example, even when all components of an IBM 560X laptop are active, the WNIC accounts for 15% of overall system energy [32]. If not optimized for power consumption, the wireless network interface card can quickly drain a device's battery [15, 45, 31, 61, 101].

To enable energy savings, WNICs are designed with multiple power modes. Generally, there are four power modes. The *idle*, *receive*, and *transmit* modes are all high energy modes which consume significant energy, while *sleep* is low-power mode and consumes nearly an order of magnitude less energy.

In low power sleep mode, most of the components in the WNIC are turned off to reduce energy dissipation. Importantly, energy saving protocols that transition the network interface card into sleep mode must know when to transition the WNIC between different power modes to avoid possible packet losses. As an example, we will examine the popular power saving mode (PSM) in IEEE 802.11 wireless LAN specification later in this thesis. 802.11 PSM transitions the network interface card between different power modes at a regular interval in order to save energy [84].

It is clear that energy efficiency is very important for mobile computing. At the same time, network performance is also of paramount significant for mobile users. To this end, how to conserve energy while still maintaining network throughput becomes an important challenge in WNIC power management. As power management is a resource management problem at the operating system level, the problem is fundamentally a tradeoff between performance and energy. Normally, high network throughput means more energy consumption and vice versa. When necessary, operating systems must be able to efficiently trade lower performance for prolonged battery lifetime.

Here, we loosely classify current power management solutions on WNIC into the following two categories:

- Transmission Power Control (TPC): Basically, TPC aims to adapting the radio transmission power (TP) along each wireless link based on the wireless characteristics of the link. In this way, different transmission powers are used to transmit different packets to minimize energy dissipation.

- Energy Efficient Communication: The basic idea is to reduce energy consumed by a WNIC by transitioning it to a lower-power *sleep* mode when data is not being received or transmitted. Energy efficient communication normally requires packets to be sent in bursts at agreed-upon intervals in order to allow the wireless client to keep its WNIC in a lower power-consuming *sleep* state for energy savings. This can be done in either wireless MAC layer or in network layer.

This dissertation focuses on network levels techniques as the key mechanism for implementing energy efficient communication. We seek to minimize the energy consumed by the wireless network interface card for a mobile device generating request/response traffic (e.g., Web browsing or TCP download) over a reliable transport protocol such as TCP.

Throughout the thesis, we investigate the tradeoffs between network performance and energy savings for several link layer power saving mechanisms such as 802.11 PSM and its optimized version, Bounded Slowdown (BSD). We propose a set of client-centered energy efficient communication techniques to support popular mobile applications. The fundamental ideas are that the client actively tracks connections, predicts when packets will arrive, shapes traffic when necessary, and keeps the WNIC in high-power mode only when necessary. Our technique allows mobile clients to save energy in a *client-centered* manner, i.e., *without* any assistance from servers, proxies or IEEE 802.11b power saving mode (PSM) in the access point [20].

Our techniques improve upon other techniques like the bounded slowdown (BSD) protocol due to Krashinsky and Balakrishnan [59] (which itself improves upon PSM with better energy savings and better round trip times by using an adaptive beacon period). First, our techniques save energy for short files such as web accesses *without* incurring slowdown. Second, our techniques also save energy for large file downloads. While small files are downloaded more frequently than large files, much more energy is consumed by large files; we believe both scenarios are important. Third, our technique requires no changes to existing hardware such as access points. Fourth, our client-side implementation can be easily deployed at individual end hosts and does not require any modifications to the network infrastructure or assistance from the remote peer. Finally, our techniques are fully TCP compliant, which allows for easy and fast deployment. We demonstrate that understanding the fundamental tradeoffs enables the better design of energy saving protocols that maintains good performance while conserving energy.

1.4 Design Principles

Before concluding this chapter, we discuss the principles that have influenced the design of the techniques developed in this thesis:

- Robustness: As power management is fundamentally a tradeoff between performance and energy, our techniques here should incur minimal overhead and performance degradation. Also, they should have strategies to support QoS adaptation and allow easy exit whenever it is not profitable to save energy.

- Extensible: While the techniques we design in this thesis are targeted to addressing WNIC energy problem for Web browsing and large TCP downloads, our solution should be easily extensible to handle other wireless communication issues in the area. For instance, our techniques should also be able to be tailored for other applications such as streaming and online gaming that have similar communication requirements. On another dimension, our client-centered approach can also be applied to solve mobility and security problems.

- Easy deployment: As protocol upgrading in the Internet is hard, our techniques should allow easy deployment through individual host upgrading without any modifications to the servers and the underlying network infrastructure. A client-centered strategy here fits the requirement perfectly. Further improvement using more aggressive solutions can be evolved from our techniques through the incremental upgrading of other network components such as routers and access points in the future.

- End-to-end principle: In any case, our techniques should keep the end-to-end semantics of a TCP connection and incur minimal additional burden to the routers and access points in the interior of the network. As demonstrated in [96], the end-to-end principle guarentees scalability and contributes to the success of the Internet.

During the entire design process, we abide by the principles stated above. Those principles are also the cornerstone of TCP and the Internet. They represent ways of solving the problem through system approaches. Our adaptation of those principles paves the way for designing successful energy efficient wireless communication protocols in this thesis.

1.5 Thesis Outline

The remainder of the dissertation organized as follows. Chapter 2 motivates our work by describing related work in the fields of wireless communication technologies and power conservation techniques. It also gives an overview of the TCP protocol stack, which is our target network protocol in this thesis. Specifically, we delineate IEEE 802.11 wireless network specifications and the related Power Saving Mode to pave the way for future discussion.

In chapter 3, we discuss our experimental methodology and describe the emulated environment we used in our experiment. We give a brief introduction to the Netfilter framework, which we used to implement our techniques. We also discuss our modification to Dummynet [95] for our experiments.

In the following chapter 4, we discuss and evaluate a technique we used to measure round-trip time. The technique exploits TCP timestamp option to infer round-trip time passively on the wireless client side.

In chapter 5, we present our energy efficient technique for web browsing. We describe key algorithms for connection tracking and connection stage transition. We discuss the benefits and potential pitfalls of reusing the RTT cache. We evaluate our technique and analyze the performance results compared to both PSM and BSD.

We describe our techniques for large file download in Chapter 6. We discuss techniques to shape traffic and compare different mechanisms for end of burst detection. We discuss our limitations and show some situations in which it is not profitable to apply our techniques. We demonstrate the significant performance benefits of our technique via extensive experiments.

We conclude this dissertation with a summary of our work and contributions in Chapter 7. Although details are only provided for web browsing and large file downloads in this dissertation, the idea is general and can be implemented in a similar manner for other mobile applications in the future. We discuss this in the future work.

Chapter 2

Background and Related Work

In this chapter, we discuss the background information relevant to our work. We also survey the related work and point out its relationship to our work. First, we give a brief introduction on wireless networking technologies. This leads to the discussion of 802.11 networks and related MAC layer techniques, including some proposed MAC layer power optimization techniques. We then present 802.11 Power Saving Mode and Bounded Slowdown (BSD) protocol. Next, we point out problems of those techniques that arises due to the mismatch between high level application requirements and the low level MAC layer specifications. Second, we give an overview of network protocol stack using TCP/IP as an example. We describe the related components of TCP/IP for this thesis, including TCP congestion control and flow control. Detailed description of TCP/IP mechanism can be referred to RFCs.

2.1 IEEE 802.11 WLAN

The focus of this thesis is the study of energy conservation techniques for wireless communications. The wireless network infrastructure we discuss in this thesis is an IEEE 802.11 based wireless local area network (WLAN). The IEEE 802.11 specification was originally proposed as an extension to wired LANs to provide last mile connectivity between the Internet and mobile hosts. From architectural point of view, it is a replacement of Ethernet in the MAC layer and physical layer in the network protocol stack with radio-related functionality. With the proliferation of mobile computing, 802.11 WLANs are widely deployed and have become the industrial standard. Today, the state-of-the-art 802.11g specification can now support up to 54 Mbps rates at much lower cost.

There are two modes in 802.11 standard for wireless networking, namely infrastructure mode and ad-hoc mode. In infrastructure mode, the wireless network consists of mobile hosts and network infrastructure components such as access points. Mobile hosts talk to the fixed access point and the communications between mobile hosts and the Internet are carried out via packet forwarding at the access point. This structure is very similar to cellular networks, in which base stations are needed to relay the packets from/to mobile stations during wireless communications.

As compared to the infrastructure mode where access points are needed, 802.11 wireless networks can also be organized to form a mobile ad hoc network (MANET) [22] "on the fly" using the so called "ad hoc" mode. In this mode, mobile hosts talk to each other without the help from the access point. Each node in the network is required to act as an access point to forward packets for other nodes so that two nodes can communicate with each other even when the distance between them is larger than the radio propagation range. In this mode, the sustainable operation of the mobile ad hoc networks is largely depends on the cooperation of each mobile host in the network. There are many algorithms and techniques proposed to discover routes dynamically and encourage cooperation in MANETs [121]. The study of power conservation techniques in MANETs is also an active research topic and will be my future research direction (see Chapter 7).

2.1.1 802.11 MAC

Based on the 7-layer OSI network protocol stack, the IEEE 802.11 standard places specifications mainly on the physical and the medium access control layers. We refer to these two layers as PHY and MAC. The PHY layer handles the actual transmission of data through a radio between nodes. It specifies a variety of RF radio mechanisms for communications such as Direct Sequence Spread Spectrum (DSSS), Frequency-Hopping Spread Spectrum(FHSS), and the most up-to-date techniques like bluetooth. It also makes provisions for data transmission rates from 4Mbps in 802.11b upto 54Mbps in 802.11g.

The functionality of the 802.11 MAC layer is to provide a set of protocols that are responsible for maintaining order in the use of the shared medium. Inherited from its wired MAC layer counterpart, the 802.11 standard also proposes carrier sense multiple access with collision avoidance (CSMA/CA) protocol. Due to the shared communication space, it is critical to ensure an unoccupied channel before transmitting data from any mobile host to reduce wireless collisions [83]. To send a packet, a host first listens to the channel for any activities. If the channel is clear, it obtains the channel and transmits the packets. Otherwise, it backs off and delays the transmission by a randomly chosen value. According to the specification, the backoff value determines the amount of time the node must wait until it is allowed to obtain the channel and transmit its packet again. Because the probability that two nodes will choose the same backoff value is small given that each node selects it independently and randomly, collisions among packets from different nodes are reduced.

The actual implementation of CSMA/CA in 802.11 is based on the RTS-CTS-DATA-ACK mechanism [114]. Whenever a mobile host want to send a data packet, it first tries to obtain the channel by sending a Request-To-Send(RTS) packet to the destination host. The RTS packet contains information of the packet length in order for other hosts to backoff properly. Upon receiving the RTS, the destination node responds with a short Clear-To-Send(CTS) packet. After the RTS-CTS exchange, the sender successfully obtained the channel, and it can safely transmit its data packet. On the receiver side, when the packet is received without any error, as determined by a cyclic redundancy check (CRC), the receiver responds with an acknowledgment (ACK) packet. A data packet exchange is regarded as successful only after finishing the RTS-CTS-DATA-ACK procedures. Compared to the CSMA mechanism in the wired case, this back-and-forth exchange is necessary to avoid the unique "hidden terminal" problem in wireless communications. In the "hidden terminal" situation, a node C does not know node A is sending data to node B when it is out of the wireless signal range of node A. As a result, it might corrupt node A's packet to node B when it attempts to send data packet to node B at the same time.

2.1.2 802.11 Beacons

We know that access points are the heart of the 802.11 wireless infrastructure networks. It is the hub between mobile hosts and the outside world. To keep the network synchronized, an access point periodically sends broadcast beacons to mobile hosts in the network. The beacon is the most important management packet in 802.11 networks. It provides the "heartbeat" of a wireless LAN, enabling mobile stations to establish and maintain communications in an orderly fashion. Because the beacon packet is a broadcast packet, every mobile host needs to listen to it periodically. The new 802.11 specification allows changing of beacon interval through the configuration of access points. Typically, the beacon interval is set to 100ms, which provides good performance for most wireless LANs.

In ad hoc networks, the lack of an access point requires one of the mobile hosts to take the responsiblity of sending beacons. After receiving a beacon packet, each station waits for the beacon interval and then sends a beacon if no other station does so after a random time delay. This ensures that at least one station will send a beacon, and the random delay rotates the responsibility for sending beacons.

As stated above, beacons broadcast heartbeat management packets. This leads to substantial overheads in generating and sending beacon packets. Despite the overhead, beacons serve a variety of functions in synchronizing the network. For example, each beacon transmission identifies the presence of an access point. Each mobile host needs to passively scan all RF channels and listen for beacons in order to find a suitable access point to associate with.

A typical beacon frame is approximately 50 bytes in length. It consists of a common frame header and the CRC field. To force all mobile stations on the network to receive and process each beacon frame, the destination address is always set to all ones, which is the broadcast MAC address.

In the following, we give a detailed description of beacon frame body. Normally, each beacon carries the following information in the frame body:

1. Beacon interval. This field specifies the amount of time between consecutive beacon transmissions.

2. Timestamp. The access point use this field to synchronize all the mobile stations in the wireless network. After receiving a beacon frame, a station uses the timestamp value to update its local clock.

3. Service Set Identifier(SSID). The SSID identifiers a specific wireless network. Mobile hosts use the SSID in the beacon to automatically configure the NIC with the proper SSID from the access point.

4. Supported rates. This field lists the transmission rates supported by the access point.

5. Parameter Sets. The beacon includes information about the specific signaling methods. (such as frequency hopping spread spectrum, direct sequence spread spectrum). For example, it contains channel number, hopping pattern and dwell time that an AP is using.

6. Capability Information. This signifies requirements of stations such as Wired Equivalent Privacy (WEP).

7. Traffic Indication Map(TIM) This field notifies a power saving mobile host whether it has data frames waiting for it in the access point's buffer.

Upon receiving a beacon, mobile hosts learn information about the particular wireless network. By extracting related information such as SSID, supported rates, and parameter sets, a mobile host can decide to join a wireless network by associating it with the most preferable access point. After joining the network, mobile hosts can still continue to scan for other beacons in case the signal from the currently associated access point becomes too weak to maintain communications. Whenever the signal drops to a threshold, mobile hosts can switch to another wireless LAN with better signal strength if possible. Currently, a multi-home strategy is proposed to enable mobile hosts to select the best wireless channel for communications. This heavily relies on receiving beacon

information. In addition, mobile hosts will abide by any changes indicated by the beacon, such as data rate, that the body of the beacon indicates.

As described above, the beacon is critical to the performance of wireless networks. There is a fundamental tradeoff in setting beacon intervals. On one hand, increasing beacon intervals reduces the number of beacons and the associated overhead. However, a large beacon interval will likely to delay the association and roaming process when mobile hosts are moving because hosts depend on beacons to scan for available access points. On the other hand, decreasing beacon intervals will make the association and roaming process very responsive; however, the network will incur additional overhead and throughput will go down. In addition, this will reduce PSM benefit because mobile hosts consume more energy in listening to beacons.

2.1.3 802.11 PSM

In this section, we discuss about how 802.11 Power Saving Mode (PSM) helps mobile hosts conserve energy. The main idea behind the 802.11 power saving mechanism is to utilize the access point to buffer packets for mobile clients and burst them to mobile clients only when polled. The access point maintains an updated record of the mobile hosts currently working in Power Saving Mode, and buffers the packets addressed to these hosts until either the hosts specifically request the packets by sending a power saving poll request, or until they change their operation mode.

To enter into power save mode, a mobile host notifies the access point of its desire by setting the power save bit in the header of each 802.11 MAC layer packet to 1. The access point receives this packet and notices the corresponding host wishes to enter power save mode. The access point then begin buffering packets for the mobile host while the mobile host transitions its wireless network interface (WNIC) to sleep mode to conserve energy.

Once every beacon period, the access point broadcasts a beacon containing a traffic indication map (TIM) to notify mobile hosts whether there are any data buffered at the access point. Since

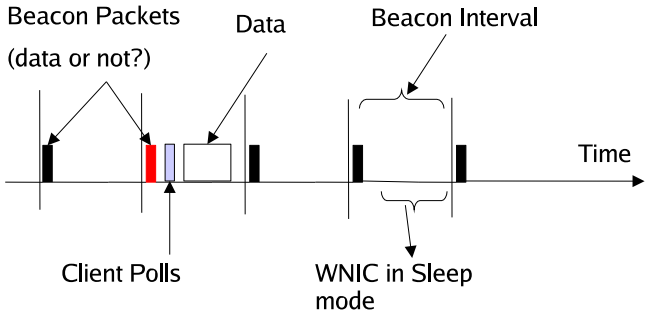

Figure 2.1: Operation of PSM. The Client wakes up periodically to receive beacons and polls the access point if there is data buffered there.

mobile hosts know the beacon interval, they can wake up to listen to beacons at a fixed frequency. When a mobile host learns from the beacon that there are packets buffered at the access point waiting for delivery, it stays awake and sends a power saving poll (PS-Poll) message to the access point to retrieve those packets. Each data packet from the access point indicates in the header of the packet whether or not there is more data outstanding. The mobile host transitions its WNIC into sleep mode only when it has retrieved all pending data from the access point. Note that the retrieval of every packet is preceded by a PS-Poll message. When the mobile device itself has data to send, it can wake up to send the data without waiting for a beacon.

Taking the advantage of modulized design, nearly all components except the timing circuit on a wireless network interface card can be shut down during power saving mode. Consequently, the WNIC consumes much less power in sleep mode. Naturally, there exists time and energy overhead for transitioning WNIC modes. Once a mobile host is in power saving mode, it transitions the WNIC to low power mode to conserve energy between beacons and transitions between modes periodically to receive regular beacon transmissions. This is clearly illustrated in Figure 2.1.

A mobile host listens to every beacon as a default setting in the 802.11 specification. In a network with a short beacon interval, this would force the mobile host to wakeup and listen to beacon more frequently, translating to a larger energy consumption for beacon packets and thus gaining less benefit from the power saving mode. As an improvement, the 802.11 specification also allows mobile hosts to skip some beacons for a short period of time.

2.1.4 Bounded-Slowdown (BSD)

The 802.11 PSM mechanism greatly reduces the energy consumptions on WNIC. However, this comes as a price of degraded performance. It is well known that TCP throughput is directly proportional to the observed round-trip time in the network [91]. Because packets might experience extra delays at the access point, the throughput of the wireless network will be reduced and the user perceived response time will be increased correspondingly. Normally, in the context of request/response network traffic, this increase in round-trip time might not be acceptable for some interactive applications such as Web browsings. For example, when a user browses a web page with a 40ms RTT, he will normally experience a short response time. However, with PSM buffering at the access point, RTT will be increased to 100ms and slowdown the browsing by more than twice with a doubled user perceived response time. Generally speaking, 802.11 PSM trades off performance for energy.

The 802.11 PSM mechanism normally operates on a fixed beacon interval with a typical value of 100ms. Although it guarantees that the RTTs are not delayed by more than one beacon period by enforcing mobile hosts to listen to each beacon, it may still be inadequate for web browsing because of the significantly increased user reponse time. Prior research [59] has concluded that a static PSM beacon interval can be too coarse-grain to yield acceptable performance during the active data transmission time. On the other handle, when mobile hosts are idle, the same value can be too fine-grained to minimize energy during the inactive period.

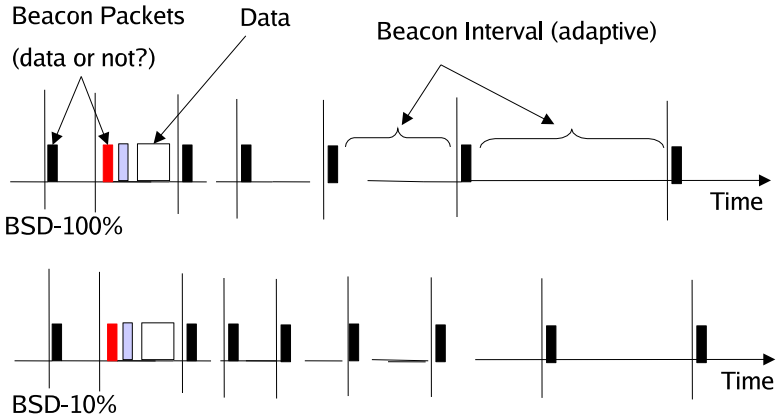

Figure 2.2: Bounded Slowdown Protocol.

As the beacon interval represents a fundamental tradeoff between network performance and energy savings for PSM, research effort has been put into investigating mechanisms for dynamic beacon intervals to better support performance and energy tradeoffs. The Bounded-Slowdown algorithm belongs to one of them and can be simply summarized as follows. After a mobile host sends any data, its network interface initially stays awke for T_{awake}. Then, it sleeps for $T_{awake} * p$ before waking up to check for and receive any buffered data. Here, p is maximum percentage increase of delays. This mechanism guarantees that no packet will be delayed more than $RTT * p$ of the original RTT at the access point. It repeats this pattern, each time sleeping for the duration since the request was sent multiplied by p. The algorithm is not affected by data being received, but it restarts whenever the mobile device sends any data (e.g., a TCP ack for data received). The behavior of PSM and Bounded-Slowdown is showed in Figure 2.2.

The BSD protocol saves energy by allowing a mobile device to skip some beacons and listen to beacons with a dynamic adjustable interval. Although the 802.11 specification already allows

mobile stations to skip beacons for a certain period of time, BSD enables the dynamic change of this period with a quantitive approach.

The implementation of the BSD protocol requires large amount of buffering at the access point since the mobile host listens to fewer beacons. The author points out that the reduced frequency of listening to beacons typically occurs when there is little network traffic from a mobile host. This normally happens during web browsing when the user is perusing the content of the web pages.

The author also concluded that fast response times are not delayed with the BSD protocol with the extra waiting after receiving beacons. The slow response times are increased by up to a parameterized maximum factor. Compared to PSM, active energy is increased since there are not many transitions to sleep mode, but the energy spent listening to beacons is decreased due to the longer sleep intervals.

In general, the BSD protocol operates solely at the MAC layer and does not require any higher-layer information. It saves energy during the inactive period, while it is ineffective during the active transmission phase.

2.2 TCP/IP Overview

As this thesis addresses energy efficient communications in the network layer, it is vital to understand how network protocols operate cooperatively to achieve data transmission. Starting from there, we begin to identify opportunities to design network protocols for energy efficient purposes.

In this section, we give an introduction to TCP, the predominant network protocol in use today. More importantly, we discuss the principles behind the congestion control, flow control, and self-clocking mechanism.

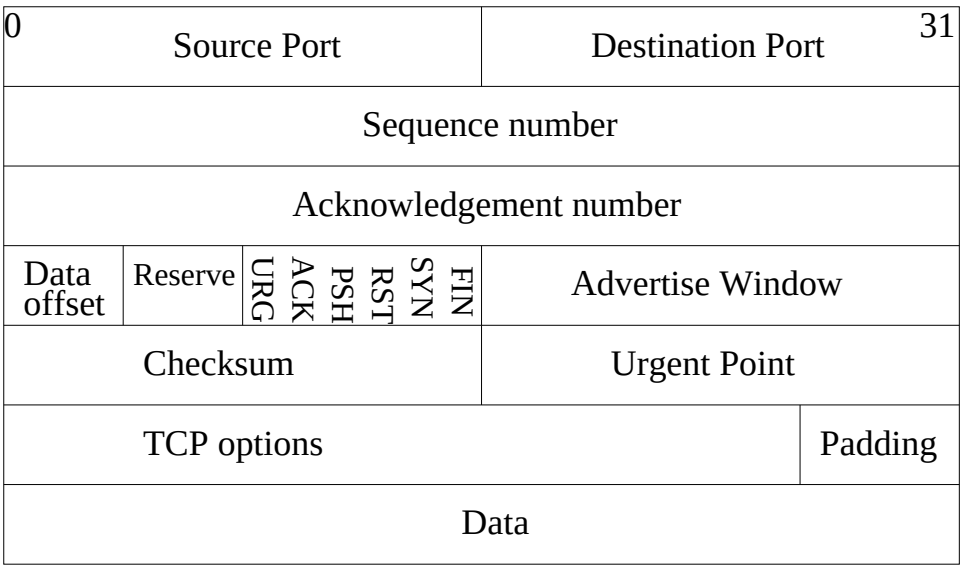

Figure 2.3: TCP Packet Header Format

2.2.1 TCP Header

The Transmission Control Protocol (TCP) [92] is the de facto standard protocol for unicast data transport in the Internet. TCP is used in a wide variety of applications and application-level protocols including the World-Wide Web (HTTP), file transfer (FTP) and electronic mail (SMTP).

TCP is a connection-oriented protocol. It provides reliable end-to-end data delivery service to the upper layer applications between any two machines in the Internet. Although a TCP connection transfers a continuous stream of bytes, in practice it does so in discrete units called TCP segments. A TCP segment consists of a TCP protocol header and a sequence of data bytes with each data byte indicated by a unique sequence number within a connection. Figure 2.3 illustrate the format of a TCP header.

Figure 2.3 depicts the TCP header format. We explain each field in TCP header in the following.

- Source Port: (16 bits) source port number.

- Destination Port: (16 bits) destination port number.
- Sequence Number: (32 bits) The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
- Acknowledgment Number: (32 bits) If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.
- Data Offset: (4 bits) The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.
- Reserved: (6 bits) Reserved for future use. Must be zero.
- Control Bits: 6 bits (from left to right):
  URG: Urgent Pointer field significant
  ACK: Acknowledgment field significant
  PSH: Push Function
  RST: Reset the connection
  SYN: Synchronize sequence numbers
  FIN: No more data from sender
- Window: (16 bits) The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.
- Checksum: (16 bits) The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. While computing the checksum, the checksum field itself is replaced with zeros.
- Urgent Pointer: (16 bits) This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment.

- Options: variable size TCP Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. The most widely used TCP options are Timestamp option, MSS option, and selective Ack (SACK) option. All options are included in the checksum. An option may begin on any octet boundary.

- Padding: variable The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

Most likely, a TCP segment from a sender is a data packet piggbacked with an acknowledgement. A TCP segment from a receiver is a pure acknowledgement. In the rest of this thesis, we refer to TCP acknowledgement packets as Acks. and the sequence number in the acknowledgement field as *Ackno*.

2.2.2 Loss Detection and Recovery

TCP provides reliable byte-stream services through an unreliable network infrastructure. Fundamental to loss detection and recovery is the TCP acknowledgement mechanism. A TCP sender is responsible for retransmitting lost packets, while a TCP passive receiver is only required to send acknowledgement packets upon receiving data packets. A TCP sender solely depends on the lack of an Ack to detect loss. This can happen in one of two ways, namely a TCP retransmission timeout, due to the lack of an Ack, or a TCP fast retransmission due to a duplicated Ack.

A TCP sender can actively detect packet loss using a retransmission timer. Whenever a TCP sender sends a packet, it starts a timer known as the retransmission timer. If there is outstanding data still not acknowledged at the time the retransmission timer expires, the TCP sender assumes that the earliest packet among unacknowledged packets have been lost and retransmits the lost packet.

Note that *Ackno* is cumulative, i.e., it always acknowledges the data that has been successfully received without any disruption in the sequence space. It has the advantages of simple and concise

because one Ack will acknowledge all packets before the *Ackno*. On the other hand, it suffers from the disadvantage that not every packet is explicitly acknowledged. To overcome the drawback of cumulative acknowledgement, selective acknowledgement (SACK) option has been proposed as a complement solution. With SACK option, the receiver can notify the sender exactly which data packets it has received with SACK blocks. Each SACK block represents a contiguous block of data that has been received successfully. The sender can know the gaps where data is missing via the SACK and can selectively retransmit the missing packets.

Besides the active retransmission timeout, the TCP fast retransmission algorithm was introduced to improve loss recovery. It is based on the assumption that packets arrive to the receiver in the same order as they were sent. Upon receiving certain number of duplicated Acks, a TCP sender believes that there is a packet loss and will retransmit the lost packet to recover from the loss quickly. Historically, this is largely true. However, with the new peer-to-peer data sharing and dissemination protocol, this feature appears to cause troubles in CDN networks [124].

Aside from packet receipt notification, TCP Acks are also used to elicit packets from the sender side, i.e., trigger the transmission of packets on the sender side. This is called TCP self-clocking. Basically, a TCP sender relies on the receiving of Acks as a signal of successful transmissions. After knowing packets left the network, the sender can continue to send out more packets into the network. This self-clocking mechanism smoothes out the transmission of TCP packets into a steady stream and reduces packet losses causes by bursting. However, this bandwidth conscious approach wastes energy on the receiver side. The techniques we proposed in this thesis convince the TCP sender to burst packets so that the receiver can save energy by transitioning to low power mode between bursts. We will talk about this in detail later in this thesis.

2.2.3 Congestion Control and Flow Control

Network congestion refers to situation when traffic exceeds network capacity. In the worst case, congestion collapse can occur in that everybody experiences extremely low throughput and the

Internet infrastructure appears to be down. TCP congestion control and congestion avoidance algorithms, first developed by Jacobson [48], operate in a distributed manner to maintain Internet operation.

To avoid flooding the network, TCP uses the "sliding window" protocol to control the number of packets it can send into the network. The "sliding window" protocol has the following advantages. First, it regulates the traffic an end host can put into the network based on the available bandwidth. Second, the sliding window protocol achieves better utilization of the available network bandwidth by allowing the sender to have up to a window of packets in the network at any time. As a result, TCP can achieve throughput that is directly proportional to the congestion window size, which in turn is virtually the available bandwidth in the network.

TCP uses a simple additive increase/multiplicative decrease(AIMD) algorithm to adjust its congestion window size. The dynamic adjusting of the window size helps to maitain the network in a steady state and rectify it if congestion ocurrs. A well tuned sliding window protocol can keep the network saturated with packets; Therefore TCP can obtain the throughput that is available from the network.

Here, we need to clarify the concepts of congestion control and flow control. Congestion refers to a situation where the network is so heavily loaded that router buffers fill up, causing packet drops. Flow control refers to the situation that a fast sender overflows the buffer of a slow receiver and so must stop and wait for the receiver. Basically, congestion control deals more on the capacities of network infrastructures while flow control is an end to end issue.

TCP uses two specific algorithms at different phases to tune the congestion window size to avoid flooding the Internet. Slow start is used at the begining of a connection to let TCP ramp up the congestion window size quickly. When a new connection starts up, the congestion window is set to one packet and slow start is initiated. Upon receiving an Ack, TCP increases the congestion window by one packet. Because each packet is acknowledged during slow start, the congestion

window grows exponentially during slow start, doubling each RTT. Beyond a threshold window size, called the slow start threshold, TCP enters into congestion avoidance phase where the congestion window is increased in an AIMD manner. Under AIMD, the congestion window is incremented linearly by one segment every RTT. Once there are packet losses, the network is assumed to be overloaded, and TCP reduces the congestion window by half and repeats the procedure to probe bandwidth.

Congestion control and avoidance algorithm prevent TCP from overloading the Internet. However, there are situations that the network is able to deliver packets but the receiver cannot absorb the packets due to limited memory buffers or a slow CPU. To avoid over-running the capacity of slow receivers, TCP uses a flow control mechanism that is integrated into the sliding window protocol. In particular, the receiver advertises its available buffer size to the sender on each Ack from the receiver to the sender. This is done using the $AdvertisedWindow$ field in the TCP header. Knowing the receiver's buffer size, the sender is then limited to its congestion window size to no more than the smaller of the network capacity and the receiver's buffer size. A TCP sender enters into persistent mode whenever the receiver side advertises a window size of zero. In this mode, TCP sender persists in sending a segment with one byte of data periodically to probe the receiver. Eventually, one of these probes will trigger a response that reports a non-zero advertised window after data is consumed on the receiver, and then the TCP sender will continue to send.

Combined, both the congestion control and flow control, a TCP sender computes an effective window that limits how much data it can send based on the smaller one of network capacity and receiver capacity. Consequently, a TCP source is then allowed to send no faster than the slowest component, the network or the destination host, can accommodate. Thus, the receiver throttles the sender by advertising a window that is no larger than the amount of data that it can buffer.

2.3 Other Energy Reduction Techniques

One important research topic in reducing energy consumption in wireless networks is the use of transmission power control to select different power levels for the sending of different types of packets. Transmission power control (TPC) has a great potential to achieve high throughput and low energy wireless communications [76].

Compared to our work, TPC is a mechanism that works purely in the MAC layer. There are many variants of TPC mechanism. Generally, they rely on the IEEE 802.11 distributed control functions(DCF) to negotiate appropriate transmission power levels for different packets, so as to maintain the desired signal-to-noise ratio [1]. A family of approaches were proposed in a similar way that varies the transmission power for different MAC layer control frames and data frames. We briefly review them below.

A typical TCP solution was presented in [26] by which an optimum RF transmission power level was selected, which assures a minimum energy expenditure for the actually transmitted bit of information for a given network configuration, channel characteristics, and packet length. Work in [27] also presents a workable approach for choosing the most energy-saving RF transmit power level, which is based on the perceived packet error rate of a single mobile host. A more refined work in [54] presents a power control MAC protocol that allows mobile devices to vary transmit power level on a per-packet basis. Work in [77] gives insight into the efficiency of those approaches and their ability to attain their design objectives. It also discusses the factors that influence the selection of the transmission power, including the important interaction between the network layer and the medium access control layers.

Ultimately, most transmission power control mechanisms are similar. However, reducing transmission power can result in energy savings, but can also result in more errors. A higher bit error rate can lead to increased retransmissions, consuming more energy. Low transmission power also reduces the transmission range and thus increases collisions due to the “hidden terminal” scenario.

In addition, the using of transmission power control may result in TCP unfairness because of the competition in wireless channel.

One body of work that is closely related to ours is PSM and BSD we discussed in Section 2.1.3. They are mechanisms that operate on the MAC layer to save energy consumption on WNIC without higher-layer information. Transmission Power control mechanisms are also work at the MAC layer. Other proposals have advocated power management at the system level [123], the application-level [32], or a hybrid of the two [106]. Our appoach in this thesis works at the network layer.

A relative complete evaluation and comparison of energy consumption of various MAC layer access protocols for wireless infrastructure networks was given by Chen et al. in [16]. They develop a framework to study the energy consumption of a MAC protocol from the transceiver usage perspective. They compare the performance of a set of popular MAC protocols using the framework.

A survey of energy efficient network protocols for wireless networks is provided in [60]. An appropriate system level or application level decision could be used to better support application specific QoS requirements when trading performance for energy.

In [101], system level power management strategies are described to turn the network interface off completely during idle periods to reduce power consumption. However, the policy does not have any guarantee on performance. Work in [13, 108] investigate an application-specific protocol for reducing the network interface power consumption for streaming media applications. The proposal uses a history-based strategy to set the sleep interval. The protocol is only applicable for application with regular access patterns such as multimedia streaming.

Kravets and Krishnan investigate the energy and delay impact of a power-aware transport protocol [60]. In their work, applications inform the protocol when the network interface should be turned off; The research investigates sleep duration and finds that a 500 ms sleep duration achieves most of the possible energy savings. They report results that reduce up to an 80% savings

compared to no power management. However, this comes at the cost of the average added delay is approximately 1 second even when no power management is used. When increasing sleep duration, the delay increases also. Overall, their approach works in a coarse-grain granularity when path RTTs are commonly within 100ms.

Examples of hardware power management are voltage-scaling processors, disk spin-down algorithms, and power-aware memory allocation. To reduce CPU energy consumption, dynamic voltage scaling (DVS) was proposed as a technique that allows processor speed to be decreased in order to run at a lower energy level (e.g., [89]). OS scheduling can then take advantage of DVS [109, 37]. The basic idea is to use fine-grain control of the clock speed to give the appearance of full computational power at peak times but less computational power at other times; this saves battery power.

In memory systems, some architectures allow individual memory banks to be powered down whenever they were not being accessed to reduce power dissipation [24, 64].

Another important area for power management is the hard disks. Like the network interface, hard disks can be disabled to save energy. To reduce disk energy consumption, many have studied disk spindown to save energy [43, 25, 112, 6, 65]; Adaptively varying the disk spin-down threshold shares similarities with adaptively transitioning between different WNIC power modes. In general, algorithms attempt to determine when there is a large time period in which there are no disk requests. However, hard disks use mechanical components and require orders of magnitude more time and energy to transition into sleep modes. Another fundamental difference with the network interface is that the information for determining when to reactivate the component may not be local to the mobile device; a packet can arrive from the network (external to the device), and the device must wake up to receive it.

Besides from the above, recent work has advocated managing energy explicitly as a resource in the operating system [123, 106, 28]. Another approach is to have the OS exploit application information cooperate with applications to save energy [32, 5].

2.4 Summary

In this chapter, we give an overview of backgound information on power savings in wireless networks. First, we present IEEE 802.11 wireless network technologies. We describe 802.11 MAC protocols and conduct a detailed discussion of Power Saving Mode with an emphasis on the 802.11 beacon mechanism. We point out performance problems with the static PSM method and introduce its optimization, the Bounded Slowdown(BSD) protocol.

Second, we discussed the upper layer network protocol architecture and TCP/IP stack. We analyze TCP header format, explain how packet losses are detected and recovered, and describe in detail the congestion control and avoidance, flow control, and self-clocking.

Finally, we briefly reviewed existing energy saving approaches on wireless network interface cards. In the next chapter, we will discuss our experimental environment and methodology.

Chapter 3

Experimental Methodology

In this chapter, we describe our experimental methodology. Given the diversity of networks and dynamically-varying traffic patterns in the Internet, it is difficult to conduct networking research in a limited environment with a high degree of confidence. Therefore, as we will discuss next, we use both emulated testbed and real Internet as well for the experiments to maximize confidence in our research results.

We begin with an discussion of our experimental methodology in Section 3.1. In Section 3.2, we describe Linux Netfilter architecture [80], our general implementation vehicle. We show our modification of Dummynet [95] to introduce round-trip time variations in Section 3.3. Finally, we present our emulated experimental environment.

3.1 Experimental Methodology

To give a full measurement, we design two basic kinds of experiments for performance study in this thesis. We run experiments on both real Internet traffic using actual servers and emulated traffic using *DummyNet* [95]. We ran two basic kinds of experiments to examine the performance of our system The emulated experiments allow us to test different parameters in a relatively controlled environment. In each of our experiments, a client initiates TCP connections to a server and requests an object from a server. In real Internet experiments, some representative servers scattered around the continent were selected.

In all our experiments, the wireless client was emulated by a 1GHz Pentium desktop machine running Linux 2.4-18 with *Netfilter*. A Dell PowerEdge server running FreeBSD 5.1-stable is configured with different RTTs using *DummyNet* to emulate Internet servers with different network characteristics. The access point is emulated using another 1GHz Dell PowerEdge server running FreeBSD 5.1-stable; we used *DummyNet* [95] to experiment with different wireless bandwidths and loss rates. In our experiments we do not differentiate between wireless and wired loss because they are the same for wireless mobile clients.

In our framework, clients transition their WNICs into *sleep* mode for periods that are bounded by the path round-trip time (RTT). Because our technique operates at a fine-grain level, it requires the use of high resolution kernel timers that have millisecond granularity. Unfortunately, the kernel timer interrupt in Linux 2.4 fires every 10 milliseconds. We patched our kernel using the KURT microsecond resolution timer [55] for our Linux client. The KURT extension to Linux provides on-demand, microsecond resolution and real-time scheduling capabilities to the standard Linux kernel. Users submit schedules of timer events, which inform the system when various real-time processes should start and stop. All communication with the KURT kernel subsystem is accomplished via a pseudo-device driver. Note that newer Linux versions are moving toward making millisecond timers standard.

Results are determined in the following way. During the experiments, we fork another process running `tcpdump` to capture packets for postmortem analysis. A simulator reads the trace and models a 2.4Ghz WaveLAN DSSS WNIC, which uses 1319 mJ/s when idle, 1425 mJ/s when receiving, 1675 mJ/s when transmitting, and 177 mJ/s when in sleep mode [103, 42]. It calculates how much time a client's WNIC has spent in high- and low-power mode so that energy can be computed. This is compared to our baseline, which is a regular TCP stream, where the WNIC remains in a high-power mode for the duration of the experiment. Note that the baseline experiment does *not* use *Netfilter*; this avoids any overhead that might be added. Also, we model the energy cost of transitioning the WNIC from *sleep* to *idle* mode as 2 *ms* in *idle* time [59]. The simulator also computes several additional quantities (such as a breakdown of wasted energy) for analysis

purposes. It is important to note that in this thesis we consider only the energy consumed during actual TCP transmissions. We do not measure energy consumption during user inactivity.

3.2 Netfilter

In this section, we give a brief introduction on the Linux kernel module programming abstraction and the netfilter architecture for our packet mangling.

3.2.1 Linux Kernel Module

Linux is a monothilic operating system. To be extensible, Linux provides a loadable module mechanism to allow developers to dynamically add functionality to the operating system without the need to rebuild and reboot the system. With on-demand loadable kernel modules, some kernel functionalities can be added into the kernel only when they are needed. This helps to reduce the kernel image size. It also reduces kernel memory requirements and simplifies hardware management. Today, most modern UNIX like operating systems all support loadable kernel module mechanism.

By definition, a Linux kernel module is software that can be loaded and unloaded into the kernel dynamically upon demand. When loaded, a module's init_module() function is called in which it registers itself to the kernel and allocates the necessary resources needed to run. Note that the module is running in kernel space and the loading of the module does not require rebuilding or rebooting the system. All the allocated resources are freed upon unloading.

The loadable kernel module mechanism is widely used for the development of device drivers. This allows hardware vendors to easily develop and test their device drivers, as well as allowing Linux to catch up with the releasing of new hardware.

The implementation of our techniques take the form of loadable kernel modules. Our module can be loaded on the fly without interrupting the system whenever a user wants to conserve energy.

It first registers itself to TCP stack and allocates packet buffers and the related data structure for connection tracking. Upon unloading, it unregisters the from TCP stack and free packet buffers by detaching its data structure from the kernel.

3.2.2 Netfilter Architecture

Netfilter is a generalized framework for packet mangling inside the Linux kernel. It is a superset of a firewall subsystem and is the basis for the implementation of IP tables and IP chains inside the Linux kernel. The Linux netfilter architecture provides a single, dedicated packet filter/mangler infrastructure that users and developers can deploy as an add-on built into the Linux kernel. Netfilter was designed to be modular and extensible.

The Netfilter architecture consists of a set of "hooks". Those hooks are well-defined points along the path that packets will be processed by protocol stacks. Kernel functions or procedures can register these hooks to participate in the processing of packets. When a packet is traversing the protocol stack, at each of these points the protocol stack will call the registered procedures to process the packet in order. Within those registered procedures, the packet can be modified, queued, or even dropped. In short, netfilter is merely a series of hooks in various points in the network protocol stack. The IPv4 packet traversal diagram is illustrated in figure 3.1.

In the following, we examine how a packet traverses through the Netfilter infrastructure. Starting from the incoming path on the left side, an incoming packet is first passed to the netfilter framework's *NF_IP_PRE_ROUTING* hook, as indicated by hook 1. At that point, routing is checked to decide where to pass the packet. If the packet is destined to the local host, it is passed to the *NF_IP_LOCAL_IN* hook (hook 2) and finially passed to the upper layer. If the packet is destined for another host, it will be passed to the *NF_IP_FORWARD* hook to continue the traversal. If the packet is unroutable, it will be dropped.

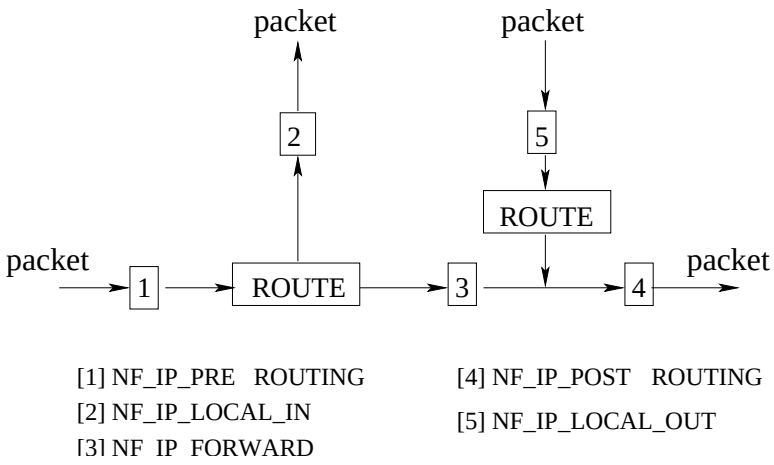

Figure 3.1: A Packet Traversing the Netfilter System

For an outgoing packet, it is first passed to *NF_IP_LOCAL_OUT* hook for examination. After filling in the related header information, the packet will then be passed to the final netfilter hook, the *NF_IP_POST_ROUTING* hook, where a route is found for the packet. The packet is then sent out.

Based on this framework, various modules have been written to provide for network firewall and connection tracking functionalities. Two representative examples are an extensible NAT system and an extensible packet filtering system (iptables) [80].

In our work, our client-centered techniques need to mangle packets. The modifications include the changing of the TCP sequence number, acknowledgement number, advertisement window size, TCP header flags and even fabricating of probing packets on the fly. We rely on the Netfilter architecture to implement them. We hook our procedures into the *NF_IP_LOCAL_IN* hook and the *NF_IP_LOCAL_OUT* hook for the processing.

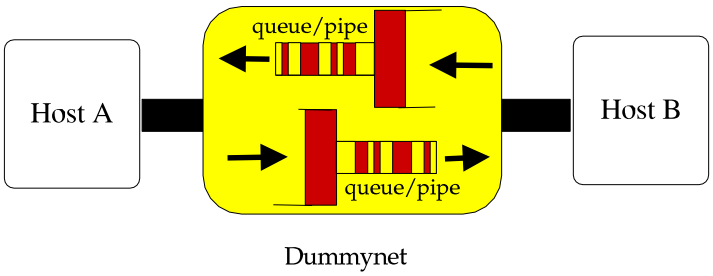

Figure 3.2: DummyNet Architecture.

3.3 Dummynet

Dummynet [95] is a flexible tool originally designed for testing networking protocols and since then has been used for bandwidth management. It simulates/enforces queue and bandwidth limitations, delays, packet losses, and multipath effects. It can be used on a workstation or on FreeBSD machines acting as routers or bridges.

Dummynet is closely associated with the firewall mechanism in FreeBSD implementations. To use dummynet, a user first specifies his interests on certain type of packets through a set of firewall rules. For example, a user can capture packets from a given protocol such as ICMP or from given ports with a set of rules. After setting up the rules, packets that match the rules will be intercepted by dummynet in their way through the protocol stack, and then passed through pre-configured queues and pipes. Dummynet queues simulate propagation delays, while pipes simulate the effects of bandwidth limitations and packet losses. With various queues and pipes to enforce bandwidth, delay, and loss restrictions to network traffic, Dummynet has been used widely to implement firewalls as well as simulate network conditions for research. Figure 3.2 illustrates Dummynet.

One of the advantages of Dummynet is its low overhead. The implementation of Dummynet is part of the TCP/IP stack and runs on the kernel without context switching costs. More importantly, there is no extra data copying between userland and the kernel. This ensures that Dummynet can process packets at a very high speed with little processing overhead and thus can emulate real networks faithfully. According to [95], dummynet is able to handle thousands of pipes with O(log N) cost, where N is the number of active pipes.

Dummynet provides the flexibility to configure each pipe and queue separately to meet different requirements. Pipes and queues can even be created dynamically. As a result, users are empowered to be able to apply different network configurations to different traffic types based on protocols, addresses, ports, and interfaces. In our experiments, we use Dummynet to emulate both wired and wireless links with different characteristics, including bandwidth, propagation delay, and loss rate.

However, there is also one defect for standard Dummynet. It can not emulate real Internet round-trip time variations with an in-order packet delivery. Dummynet only allows a fixed delay associated with pipes and queues. However, one can create multiple links with different delays between the sender and the receiver. This will cause out-of-order packet delivery at the client because packets transmitted later with lower delay can overtake packets transmitted earlier with higher delay. The out-of-order packets will trigger TCP fast retransmissions, which is an undesired effect in this situation because packets are not lost. As a result, TCP will suffer from out-of-order packets if standard Dummynet is used to emulate round-trip time variations.

To avoid out-of-order delivery while still varying round-trip delays, we modified Dummynet in a way such that packets cannot reach the destination until the packet transmitted earlier has arrived. This enhancement enables us to emulate real Internet dynamics that was not provided by dummynet. We use this enhanced feature in all our emulations.

In the following, we describe our implementation of this enhancement. Our modification works as follows: instead of assigning each pipe a fixed delay, we associate a pipe delay array with each

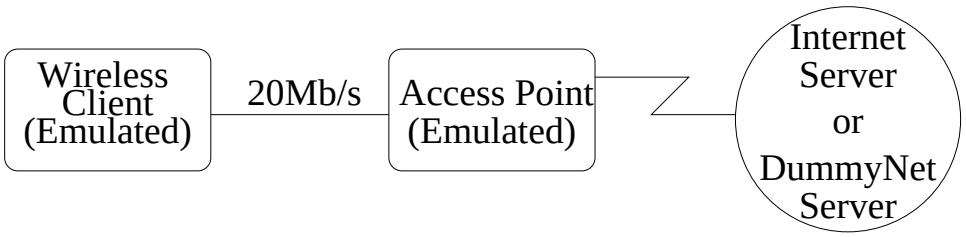

Figure 3.3: Our experimental setup with dummynet and tunneling.

pipe. This is an extension to the original configuration with one delay per pipe. The number of elements in each array is adjustable based on the user's specification. The more the bins, the more accurrate the emulation will be. For every N packets, we dynamically choose a delay d from the pipe delay array and the following N packets transmitted through the pipe afterwards will experience d delay before reaching the next hop.

Many previous studies in Internet dynamics have pointed out that round-trip time variations conform to a shifted gamma distribution with heavy tails [70]. We model round trip time variation using an uncorrelated gamma distribution. We obtained this distribution by performing ping tests, gathering several thousand samples from the `ftp.cs.washington.edu` sever, (which has a 61 *ms* base RTT), and then using these samples to determine the parameters to the gamma distribution. Note that while a realistic distribution is likely correlated, (1) *DummyNet* itself cannot handle such a distribution, and (2) an uncorrelated distribution is *more* difficult for our system to handle effectively, due to the increased unpredictability of round-trip times. To simulate peak traffic on round-trip times other than 60 *ms*, we scaled this gamma distribution proportionally to the new round-trip time.

3.4 Tunneling and Experimental Environments

Figure 3.3 shows our emulated experimental environment. We use IP tunneling to bridge IP traffic between the emulated access point and the emulated wireless client.

Tunneling is a method of using an internetwork infrastructure to transfer a payload, such as packets. The packet is encrypted and encapsulated with an extra header generated by the tunneling protocol. This extra header provides routing information between the two ends of a tunnel. The encapsulated packet is routed between the endpoints over the transit internetwork. At the destination, the packet is de-encapsulated and forwarded to its final destination.

The most popular tunneling protocols used to create VPNs are Point-to-Point Tunneling Protocol (PPTP), Layer 2 Tunneling Protocol (L2TP), IPsec, and IP-in-IP (IP-IP). Packet structure after tunneling consists of the outer IP header, the tunnel header, the inner IP header, and the original IP payload itself.

In our experiment, we use IP-IP tunneling to set up a tunnel between the client running Linux 2.4.18 to a FreeBSD box running Dummynet as access point, and then routing to the outside world. With dedicated tunnels between the emulated access point and the wireless client, we were able to isolate the "wireless" traffic from other interference in the network, making our results more accurate. More importantly, tunneling allows us to experiment with various wireless bandwidth and loss rate between the emulated access point and the wireless client.

3.5 Summary

In this chapter, we described our experimental methodology. We give a brief review on Linux kernel module mechanism and an extensive discussion on Linux Netfilter framework.

We also gave a detailed discussion on Dummynet, which helps to build the entire experimental environments for this dissertation. We discussed our enhancement to Dummynet that allows the emulation of real Internet RTT variations with an in-order packet delivery. We explained how to configure Dummynet with different settings to simulate various network traffic characteristics in order to quantify the effects of our techniques. Finally, we illustrate how we use IP tunneling to establish a virtual tunnel between the wireless client and the access point for our experiment setup.

In the next chapter, we will discuss a technique we use to estimate round-trip time on the wireless client side.

Chapter 4

Inferring Round-Trip Time

In this chapter, we describe and evaluate a technique that exploits the TCP timestamp option to passively estimate round-trip time at an arbitrary measurement point along a path. This is one of the enabling techniques to our client-centered energy efficient communications. We use this technique at the client side to estimate the round-trip time to direct our transition between WNIC modes.

We first give an introduction to the technique, then we discuss some similar techniques for round-trip time measurement. We describe our implementation in detail and present our performance results.

4.1 Introduction

Path RTT plays a central role in the behavior of a TCP connection. For example, previous research has shown that TCP throughput is inversely proportional to path RTT. For network administrators, RTT reveals the network topology [40]. Path RTT can also provide an insight on network congestion along the path [125]. Additionally, the recent development of overlay networks and peer-to-peer applications relies heavily on path RTT for the correct operation of their services [94].

There are many techniques to measure path RTT. One is to actively send periodic probe packets for the measurement [107]. One passive estimation method is to use the SYN/SYN-ACK interval and slow start traffic as the sole RTT measurements [53]. Another passive method infers the path RTT based on the simulated congestion window size [51]. Unfortunately, these methods are insufficient to realistically capture RTT variation along a path because of the inadequate

number of samples. RTT variation reflects network congestions and other end-to-end dynamics along the path. Capturing RTT variation provides insight to network administrators on network configuration issues such as queue management and buffer provisioning.

Although one can associate data segments with their ACKs by matching the sequence number with the acknowledgement number at the measurement point (assuming no packet loss), the reverse association of data segments to the ACKs that triggered them is particularly hard because of TCP's sliding window mechanism [67]. In addition, packet loss and out-of-order packets at the measurement point complicate the problem because the disadvantage of TCP's cumulative acknowledgement scheme.

In this section, we present a novel passive measurement method that exploits the *TCP Timestamp option* to measure path RTT at an arbitrary measurement point. The TCP timestamp option is a performance enhancement extension for the precise estimation of RTT in a TCP transmission. When each TCP segment contains TCP timestamp values, a passive measurement method can take advantage of those timestamps to obtain accurate path RTT measurements. According to the TCP timestamp definition, a sender places a timestamp in each data segment, and the receiver reflects these timestamps back in ACK segments. By locating and pairing those timestamps at the measurement point, we estimate path RTT in a similar way as TCP itself.

Our technique makes three important contributions. First, our method reports an *accurate* RTT measurements for a TCP connection. Our method can produce accurate measurements because the TCP timestamp option itself is designed to help accurately estimate RTT. We are able to obtain samples throughout a connection's lifetime. Second, our measurements provide insight on Internet end-to-end delay dynamics [70]. With the accurate measurements of path RTT, other important connection characteristics such as congestion window size, retransmission timeout value, and available bandwidth, can be inferred more accurately [49]. Those characteristics can then be used to obtain a comprehensive knowledge of network congestion characteristics. Finally, our work shows that TCP timestamp option can be a benefit for network administrators for passive

measurement. This is an argument for a wide deployment of the *TCP Timestamp option* in the Internet. We believe that a better design of *TCP Timestamp option* can be more beneficial in the future.

In the following sections, we give the basic idea of TCP timestamp options and describe our algorithm and provide implementation details. We verify our method in a controlled emulated environment with extensive experiments.

4.2 Related Work

There are two broad ways to measure path RTT, namely active method and passive method. Active methods periodically send probe packets to infer path RTT [107]. The problem with active methods is that they inject extra traffic on the connection and they can only report RTT for a given path for a specific probe.

On the other hand, passive methods work by capturing packets on a given connection at a measurement point and infer the path RTT based on these packets. One passive method depends on special packets in connection setup and slow start phase to infer path RTT [53]. Another passive method in [51] attempts to infer the congestion window size on the sender and uses the window size to compute path RTT. The drawback with these techniques is that while they do not inject any traffic, they suffer from either relatively few available samples or inaccuracy due to uncertainty in the association of packet pairs. In contrast, our passive technique obtains a much larger number of accurate samples over most connections.

Although one can associate data segments with their ACKs by matching the sequence number with the acknowledgement number at the measurement point (assuming no packet loss), the reverse association of data segments to the ACKs that triggered them is particularly hard because of TCP's sliding window mechanism [67]. In addition, packet loss and out-of-order packets at the measurement point complicate the problem because the disadvantage of TCP's cumulative

acknowledgement scheme. As a result, it is challenging for passive RTT measurement methods to be accurate.

4.3 Overview

Most passive path RTT measurement methods infer RTT based on pairing of packets that cross the measurement point. Although one can associate data segments with their ACKs by matching the sequence number with the acknowledgement number at the measurement point (assuming no packet loss), the reverse association of data segments to the ACKs that triggered them is particularly hard because of TCP's sliding window mechanism [67]. This makes it hard to collect more than a handful of samples, typically restricted to connection setup and slow start.

The basic idea behind our passive technique is to use timestamps instead of only sequence numbers in order to associate packet pairs. The timestamps allow us to, as opposed to previous techniques, obtain accurate samples throughout the lifetime of the connection. We assume here that the measurement point can observe traffic in both directions on the connection.

The TCP timestamp option was defined in [46, 47] as an extension intended to estimate RTT on TCP senders more accurately. When both hosts, H_1 and H_2, on a TCP connection agree to use timestamps, each packet sent from H_1 to H_2 contains two items: the current time on H_1 when the packet was sent and the timestamp of the most recently received packet from H_2. The other direction is similar. A detailed description of the TCP timestamp option can be found in [47, 46, 113].

As discussed above, our technique relies both end hosts using the TCP timestamp option. Accordingly, we conducted a measurement study that consisted of probing 500 web servers previously used by Floyd to determine their type of TCP implementation. We found that 92% of those

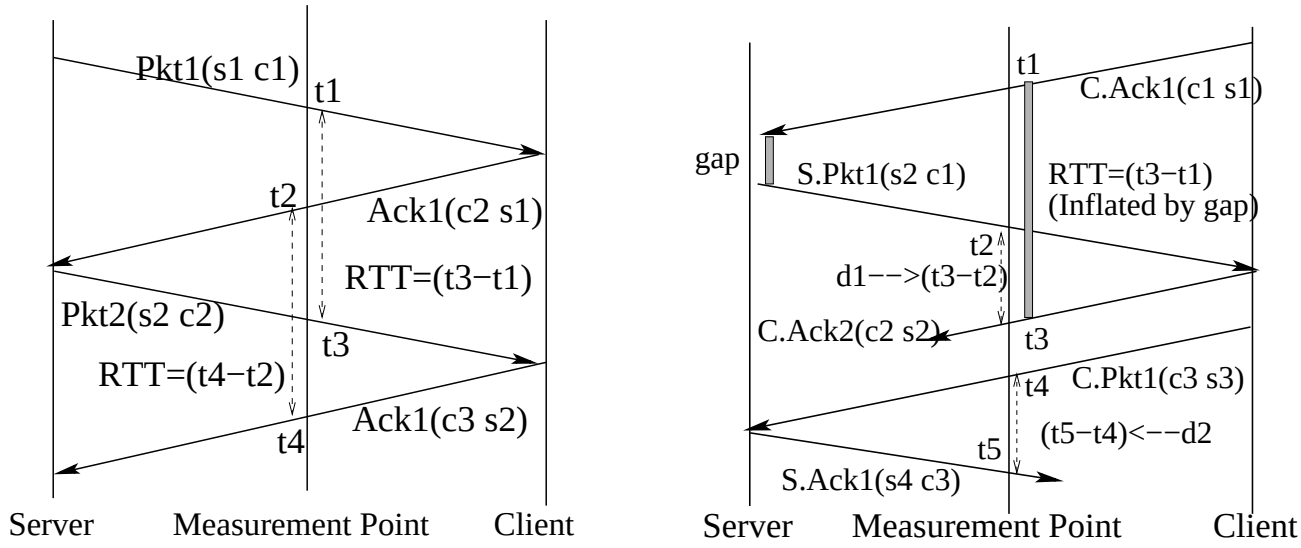

Figure 4.1: Estimating the RTT with timestamps. On the left is the basic idea, and on the right is the problem posed by gaps on the sender.

servers support the TCP timestamp option. In addition, most current operating systems also support the timestamp option. As the use of timestamp has increased in recent years, we believe the percentage of servers that support the *TCP Timestamp option* will likely increase in the future.

4.4 Our Technique

This section describes our implementation of a new passive RTT measurement using TCP timestamps. First, we give the basic idea, and then we discuss potential sources of error.

4.4.1 Basic Measurement Method

Our basic technique is to use TCP timestamp information to construct tuples consisting of a data packet, an acknowledgement, and the next data packet triggered by the acknowledgement. Once we have such a tuple, we can compute the round trip time based on the time the first and last packets in the tuple crossed the measurement point. Associating these packets without timestamp

information is a difficult problem, because associating a data packet with the acknowledgement that triggered it may require knowledge of the congestion control window at the sender side (information that is typically not available to the measurement point). Note that RFC 1323[46] only requires that the *Timestamp Echo Reply* carry the *Timestamp Value* of the most recently received segment, rather than the *Timestamp Value* of any pure acknowledgements received. In practice, however, most TCP implementations do echo back the *Timestamp Value* of pure acknowledgements, which enables our technique.

Figure 4.1 illustrates our measurement method. We record both the arrival time and timestamp value pair $<$ *Timestamp Value*, *Timestamp Echo Reply* $>$ for every packet that crosses the measurement point.

First, $Pkt1$ carries timestamp values $< s1, c1 >$ from the server, and crosses the measurement point at time $t1$. Second, $Ack1$ is the response for $Pkt1$ and carries the timestamp pair $< c2, s1 >$ in which the *Timestamp Value* from $Pkt1$ is $Ack1$'s *Timestamp Echo Reply*. This allows the measurement point to associate $Pkt1$ with $Ack1$. Third, $Pkt2$, which is triggered by $Ack1$, carries the timestamp pair $< s2, c2 >$, and crosses passes the measurement point at time $t3$. Note that $c2$ is the client side timestamp of the $Ack1$, echoed by the sender. This allows the measurement point to associate the data packet ($Pkt2$) with the acknowledgement that triggered it. We now have an association between $Pkt1$ and $Ack1$, and another association between $Ack1$ and $Pkt2$, with $Ack1$ as a common element. These two associations are combined to form the triple $< Pkt1, Ack2, Pkt2 >$, comprising a single round trip sampled as $t3 - t1$. The association of a data packet and its acknowledgement can be made by examining the sequence numbers provided in the TCP headers. However, the reverse association is impossible to make at the measurement point by sequence number alone, because an acknowledgement triggers a new data packet based on the sender's congestion window rather than the next sequence number.

We obtain another RTT sample ($t4 - t2$) when $Ack2$ arrives at the measurement point. This sample is based on the associations between $Ack1$ and $Pkt2$, and $Pkt2$ and $Ack2$, sharing $Pkt2$

as a common element. Note that we are able to make these associations because a packet that is triggered by an acknowledgement will carry the *Timestamp Value* of the acknowledgement as its *Timestamp Echo Reply*. Similarly, an acknowledgement for a received packet carries the *Timestamp Value* of the packet as its *Timestamp Echo Reply* (as shown in Figure 4.1).

Our calculation is similar to TCP's RTT estimate algorithm when receiving ACKs. We produce the same value as estimated by TCP when there is no delay in terms of timestamp value between the arrival of ACKs and the sending of packets. This is generally true in bulk data transfer applications where the TCP sender is continuously emitting data. Note that we did not compare timestamp values across machines and thus we need not worry about timestamp synchronization problems during our estimation.

4.4.2 Potential Error

Inflated Estimation One possible source of error with our technique is that our measurement could report a value that includes gaps on the sender. Retransmission delay, context switching, and application level delay can all cause such gaps. In the example shown in Figure 4.1, our RTT estimate ($s2 - s1$) includes delay at the sender. As our goal is to accurately measure the RTT, we would like to eliminate samples that include sender gaps. We have designed a simple, yet effective, filter to exclude such samples.

Data packets should be acknowledged by the receiver quickly, and thus the packet-acknowledgement measurement should not include sender gaps. Note that the cost of delayed acknowledgement is included in most RTT measurements, so we also include this in our measurements. Pure acknowledgements, on the other hand, may not require any action when the acknowledgement is received. Therefore, we keep track of the *maximum* time between a data packet and its associated acknowledgement crossing the measurement point, for both sides of the connection. Value *d1* indicates the maximum RTT between the measurement point and client, and *d2* indicates the maximum RTT

between the measurement point and server. The intuition here is that $d1 + d2$ represents our idea of the upper bound on the RTT. Our filtering rule is that any estimation that is bigger than the sum of the maximum values *d1* and *d2* over the entire stream is considered an inflated estimate and thus is discarded. Note that while we applied this filter postmortem, it could also be applied on the fly.

Assuming the right-hand side of Figure 4.1 showed the entire transmission, then distance *d1* and *d2* are $t2 - t1$ and $t4 - t3$, respectively. In this (short) transmission these would also be the maximum values, and their sum would represent our idea of the upper bound on RTT. Any measurement (most likely $t3 - t1$) larger than that sum is discarded. Note that the calculation of *d1* and *d2* must be based on the time differences between data segments and their acknowledgements, as these measurements are the least likely to include gaps. We again use the *Timestamp Value* and *Timestamp Echo Reply* in determining when to include a distance calculation, as this pairing will essentially ignore measurements based on loss. The alternative is to use sequence/acknowledgement number to pair [53], which is cumbersome.

TCP Timestamp Granularity Typical TCP timestamp granularity is $10ms$ for most Internet hosts. As a result, some packets will carry the same *Timestamp Value* if they are sent close together (e.g. within 10 *ms*). This introduces potential errors into the packet association. To counter this, we take RTT measurements only on the *first* segment with a new *Timestamp Value*. Despite this, we are able to produce samples throughout the lifetime of the connection, and our technique is sufficient to capture RTT variation. Note that extremely small RTTs (e.g. less than the granularity of both end-point clocks) are a potential difficulty. We are currently investigating techniques of dealing with this situation.

Loss and out of order packets In TCP, when out of order packets arrive, the timestamp value echoed by the reciever is that of the most recently *accepted* packet, rather than the timestamp value in the new packet. For this reason, our technique will discard the sample based on the fact

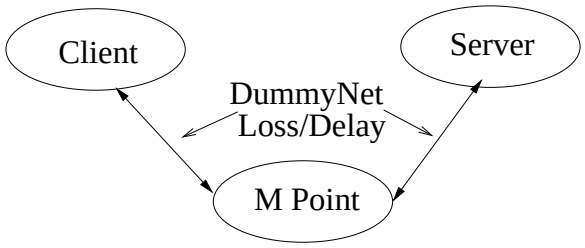

Figure 4.2: Experiment Environment for RTT Measurements

that it is a timestamp value we have previously used. This also prevents collecting a sample when loss occurs.

4.5 Evaluation

In this section, we evaluate the accuracy of our passive measurement tool. We choose three representative applications to run on the upper layer, bulk data TCP download, web browsing, and interactive telnet. We believe those three types of application cover the majority of TCP based Internet applications. We present a detailed, quantitative analysis of our method.

4.5.1 Evaluation Methodology

Figure 4.2 shows our controlled, emulated experimental environment. We use Dummynet to emulate different network conditions between the server, the measurement point, and the client.

We vary RTT on the server with a heavy tail shifted Gamma distribution [2]. The server's timestamp resolution is 1ms and the client's timestamp resolution is 10ms in all our experiments.

We introduce losses on all four paths on both sides of the measurement point. We extend the server's TCP stack to log *TCP RTT* estimates whenever it was generated on receiving a segment. We compare the *TCP RTT* estimations with our measurements point by point. Because it is

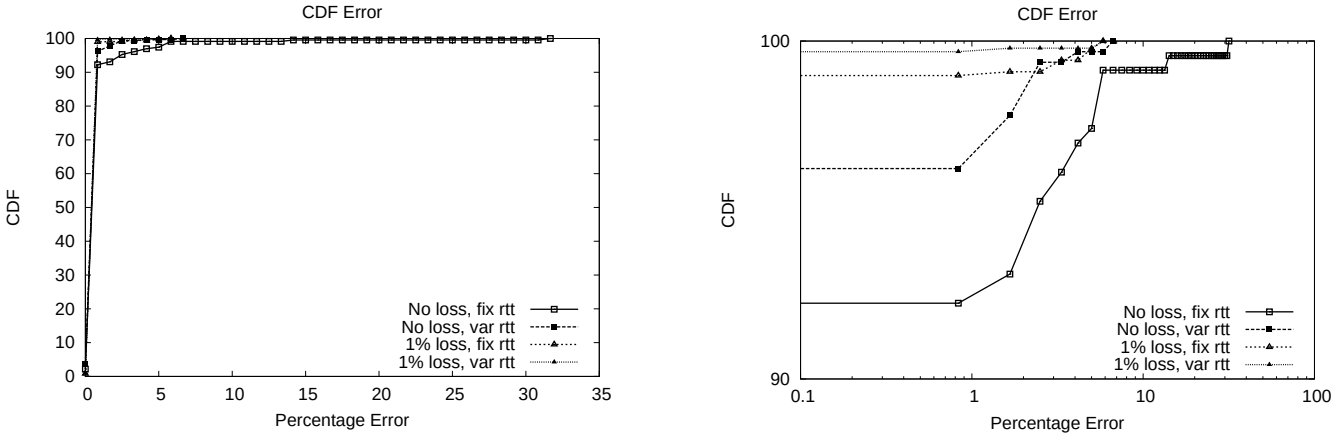

Figure 4.3: Error CDF of ftp download with different network conditions (fixed/variable RTTs and no loss/loss). In the second graph, the y-axis starts at 90%.

Loss	RTTs	samples	Max *d1 + d2*	Max % error	Avg % error	Filtered
0%	120*ms* Fixed	247	220ms	32%	1.1%	1
0%	120*ms* Variable	334	206ms	5.8%	0.8%	0
1%	120*ms* Fixed	933	231ms	6.7%	0.8%	6
1%	120*ms* Variable	1018	221ms	5.8%	0.7%	2

Figure 4.4: Summary of results from ftp download.

possible for several segments to have the same *Timestamp Value* when timestamp granularity is large, we only take RTT measurement on the first segment with a new *Timestamp Value* during our measurement to eliminate errors due to the association uncertainty for those packets. We still produce a large number of samples because we take one measurement approximately 10ms and it is sufficient for catching the RTT variation.

4.5.2 FTP and HTTP Downloads

Figure 4.3 shows our measurements on a ftp download of a 3MB file from server to client with different network configurations. Our passive RTT measurements match the TCP estimates with a high degree of fidelity under all network conditions. Figure 4.4 shows a detailed analysis of the data. Because the client timestamp has a 10ms granularity, for ftp we obtain approximately one measurement every 10ms. More than 96% of our samples are within 1%. Most of the rest are within 8%, with only one outlier (32%). Further investigation revealed that this outlier was caused by a small FTP request processing delay. The average error is approximately 1%, which indicates that our measurements are accurate for bulk data transfer applications. This includes the cases when there is variation in RTTs. Note that previous passive measurement methods, which depend on connection setup and slow start phase to infer RTT, would generate less than 10 samples for these downloads. Unlike our method, this would certainly fail to capture RTT variation.

In addition, our measurements are accurate when packet loss occurs. In fact, with a lossy network, we obtain more measurements because the congestion window size is smaller—meaning that fewer samples are discarded due to clock granularity. This is despite the fact that some samples are themselves discarded due to loss.

There are two things we note about our filtering scheme. First, it successfully helps us remove inflated samples. This includes measurements that were inflated by a one second retransmission delay due to server time out. However, we cannot detect delays smaller than our maximum *d1* + *d2*, which is what occurred with the case of 32% error—our measurement technique cannot account for arbitrary delay in sending an acknowledgment packet. Second, we see that our filtering rule in most cases avoids filtering out valid large RTTs. In particular, only one valid large RTT measurement was mistakenly removed as an inflated sample in our experiments.

Figure 4.5 shows results on an HTTP download over the different network conditions. For this experiment, we have a relatively small number of measurements because web pages are typically small. The average error overall experiments is 1.1%, and the worst-case error is 9%.

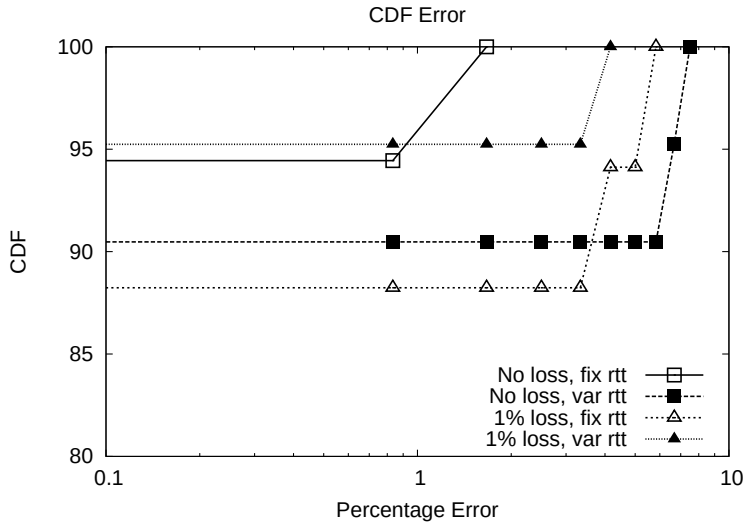

Figure 4.5: Error CDF of http download with different network conditions (fixed/variable RTTs and no loss/loss). Note that the y-axis starts at 80%.

4.6 Summary

In this chapter, we presented a passive measurement technique for round-trip time estimation using the TCP timestamp option. We evaluate the method in a controlled, emulated experiment environment with different network configurations and upper layer applications.

The method uses TCP timestamp options to associate packets and compute RTT as the differences between timestamp values. Our method uses a similar method as TCP to compute RTT and report accurate RTT measurements compared to TCP's estimates. The evaluations show that the average errors are below 1ms for FTP downloads and web browsing. The method can also faithfully observe RTT variations along the path. Our evaluations show that our timestamp based technique is accurate in measuring path RTT for a variety of applications. Our measurements can observe path RTT variation and report accurate RTT measurements even for connections with high loss rate and variation. Our technique is most accurate for bulk data transfers, which is what

we use it for in the following chapters. We used simple, but effective, filtering rules to detect server gaps and discard the resulting incorrect samples. Overall, we believe that as timestamps become almost universally used, our technique will be a significant advance in passive RTT measurement.

We apply this technique at the client side to estimate round-trip times to direct the transition of WNIC between different power modes. In the next chapter, we will discuss our implementation and experiments with energy efficient web browsing.

Chapter 5

Energy Efficient Web Browsing

The goal of our research is to investigate network-layer, energy-efficient wireless communication techniques for well-known mobile applications. In principle, energy efficient wireless communications depend on energy-saving methods that trade-off download time for energy. With the awareness of upper layer application characteristics, network-layer, energy-saving methods can better support fine-grained QoS when trading performance for energy.

In this chapter, we investigate our techniques for one popular wireless application, namely web browsing. We begin with a discussion on web traffic characteristics. We then present implementation details including our connection table as well as how we track connections. We finish with an analysis of performance results.

5.1 Web Browsing and HTTP

The Web is simply a wide-area client-server system. Web servers are repositories of information, and clients access this information from servers through the Internet. In Web, a client can either directly communicate with a server or do so indirectly via intermediate agents called proxies. Normally, the client and the server use Hypertext Transfer Protocol (HTTP) in the application layer to exchange information.

HTTP is the application level protocol underlying the Web. It defines the request-response interaction between clients and servers. HTTP is the predominant protocol for the Web today. The current version of HTTP is 1.1, also known as Persistent-HTTP (P-HTTP), which supports many enhanced features including persistent connections, pipelining, and chunked transfers. In general,

HTTP can be layered on top of any transport protocol that provides reliable data delivery service. However, in practice, it is most commonly layered on top of TCP, as TCP is the predominant transport protocol in the Internet.

The key ideas behind P-HTTP are to use a persistent TCP connection for the transfer of multiple Web page components and to pipeline the transfer of these components. With persistent HTTP support, multiple HTTP sessions can share the same underlying TCP connection and thus reduce the overhead of establishing separate TCP connections for each HTTP session.

HTTP is designed to be stateless, which means that there is no state information carried from one HTTP request-response interaction to the next. The stateless property of HTTP simplifies the protocol. HTTP Cookies are introduced to keep minimal state information.

HTTP messages are passed in a format similar to MIME, A request message from a client to a server typically contains the following information:

- Method: indicates the action to be performed. HTTP GET and POST are the two most commonly used methods in the Web. GET is used to retrieve web objects while POST is used to append data to a resource.
- Universal Resource Identifier (URI): specifies where the web object is located. Depending on the web object type, a URL could point to a static HTML page or a dynamic object that is generated at the server side based on the client's inputs.

Upon receiving HTTP requests from the client, a Web server arranges the transfer of the requested objects into HTTP response messages to the client. A HTTP response typically contains the following:

- Status: indicates whether the request is successful or not.
- Entity header: specifies meta information pertaining to a web object, including the content encoding, content length, content type, last modification time and expiration time.

- Entity body: the content of the requested object encoded as specified in the entity header.

A Web page is usually composed of multiple components such as text, images, video, audio, etc. Those components are called embedded objects of a web page. A user's Web browsing is often in the form of browsing sessions. During a browsing session, the user typically uses a web browser to download the entire Web page from servers, including all the embedded objects. The web browser then renders the contents into a user readable web page.

A typical web browsing works as follows: After reading the URL typed by the user, a web browser first sends out HTTP GET method to retrieve the HTML file corresponding to the Web page. It then parses the HTML page to determine which components, if any, are embedded in the page. Finally, it retrieves each of the embedded components using concurrent HTTP sessions. Although not necessary, the components of a Web page usually reside on the same server as the HTML file. This provides the opportunity to use P-HTTP to retrieve the objects. The browser renders the components in a manner that is convenient for the user to read. For instance, image components are displayed on a screen while audio components are played on an audio device. The browser may invoke a separate helper application to render components of data types that it is unable to process.

At the TCP layer, the timeline for a typical Web page download as follows: First, the client sets up a TCP connection to the server with a three way TCP handshake. Following that, HTTP messages are exchanged through the TCP connection to download web objects. In case more web objects need to be transferred (e.g., embedded images), a separate HTTP request-response exchange is used for each such transfer. Whether this HTTP message exchange takes over on the existing persistent TCP connection or on a new separate TCP connection depends on the HTTP protocol and where the embedded web objects are. If both the web browser and the web server support P-HTTP, the TCP connection is not torn down after a HTTP session is finished. Otherwise, either the web server or the web browser will send out FIN packets to shutdown the connection.

Previous studies have already showed that web traffic tends to be bursty. This is because individual Web objects tend to be small in size with an average page size of 26-32 KB [69]. Each

Web object download constitutes a burst. As HTTP sessions are short in time, the bursts tend to be short in length.

Due to its bursty natural, Web traffic has many "silent intervals" where no packets are expected for a period of time. Work in [59] already showed that the wait-for-server and inactive periods during Web browsing present opportunities for the network interface to enter a sleep mode. Our energy efficient technique exploits those intervals to conserve WNIC energy.

The basic idea behind our technique here is to predict when packets will arrive, keeping the WNIC in low-power mode in those intervals. This is achieved by maintaining and tracking the state of each client-initiated connection during web browsing. When *all* connections are idle, i.e., not actively receiving or sending data, the client transitions the WNIC to a lower-power *sleep* mode. The client then transitions the WNIC to back to high-power mode before the next packet (on any connection) arrives. Our technique uses round-trip time estimates to determine when to transition the WNIC between *idle* and *sleep* mode. Our technique is implemented with no change to TCP and the web server. Our solution saves energy without increasing transmission time because it does not change the web traffic.

5.2 Energy Efficient Techniques

In this section, we describe the design and implementation of our client-centered techniques for Web browsing. We assume that when the client wishes to save energy for web downloads, it informs the client OS of its intentions. The client OS then must restrict the kinds of network traffic that can be sent and received. This is further discussed in Section 5.2.9; We assume that the client OS will transition the WNIC into high-power mode when data is sent from the client, and we inform the client OS to transition the WNIC between high- and low-power modes based on our algorithm. for the rest of this chapter, we consider only HTTP traffic.

Ideally, the client would only be in high-power mode while packets are actually arriving and in *sleep* mode at all other times. In practice, the client makes predictions about when packets will arrive, transitioning the WNIC to high-power mode if it expects packets and *sleep* mode if not. The prediction of the arrival time of incoming packets is nontrivial even when the client has only one outstanding connection. This is particularly true when packets are delayed or lost in the network. Furthermore, the existing Internet introduces many factors that do not exist in simulated networks. These factors contribute to delay and loss in unpredictable ways.

In a client where multiple concurrent connections exist, it is even more challenging to predict the arrival time of the next packet because of accumulated error over many connections. We do two things to solve this problem. First, we track every concurrent connection in the client, collecting detailed information and making predictions. Second, we cache round-trip time information about each site visited by the client to allow us to save energy during connection setup. We discusses each of these techniques in more detail in the following sections.

5.2.1 Connection Tracking

The client tracks each of its initiated open connections. This provides all the necessary information to predict (1) when to transition the WNIC from high to low power mode and (2) how long to keep the WNIC in low power mode. The client divides an individual connection into stages, between which transitioning the WNIC to *sleep* mode is possible; each stage is the client's estimate of the server's TCP window. Stages are easier to distinguish during TCP slow start. Fortunately, web objects are generally small, so many HTTP connections spend a significant percentage of time in slow start. The client builds a connection table to hold detailed information about each connection, which allows accurate prediction of the next packet arrival time. We first discuss tracking a single connection and then discuss combining information from many connections to determine when to transition the WNIC.

Id	*Status*	*Site*	*Stage*	*Num Packets*	*Next Stage*		*Current Expiration*	*SRTT*	*Var.*
					Start	*End*			
1	*idle*	www.cnn.com	7	—	140	160	—	50	5
2	*active*	www.espn.com	3	6	232	TBD	102	143	12
3	*idle*	www.cnn.com	12	—	147	174	—	50	5
4	*finished*	www.cnn-ads.com	—	—	—	—	—	20	5

Figure 5.1: Sample connection table with 4 concurrent connections (two *idle*, one *finished*, and one *active*). *TBD* means that the value has not yet been determined.

5.2.2 Connection Table

The key data structure used in our system is the *connection table* (shown in Figure 5.1). The connection *status* field reflects four possible connection states: *active*, *idle*, *finished* and *saturated* (discussed below). The *site* field is used to index into the site table maintained on the client to locate the detailed information about the remote server site. (The actual table uses IP and port number. For presentation purposes, we present the logical name.) We partition each connection into *stages* and keep the stage number in the *stage* field. We also record the number of packets received in each stage. The next stage *start time* and *end time* are the predicted starting and ending times of the next transmission stage. We use the current estimate of the round trip time (in the *SRTT* field) and variance (*var*) to compute the start and end times of the next transmission stage. If the connection is *active*, we also keep track of when the current stage is expected to end (*current expiration*). We use a timer to detect when there are no packets received within a threshold amount of time (described below); this is a possible indicator of the end of a stage.

In the example connection table shown in Figure 5.1, there are 4 concurrent connections. This is a sample of a connection table taken at time 100. The second connection (marked *active*) is receiving data, the first and thirdconnections are between stages (marked *idle*), and the fourth connection is *finished*.

5.2.3 Next Stage Prediction

Whenever stage i ends, the client predicts the stage $i+1$ start and end time as $T_{first} + SRTT - VAR$ and $T_{last} + SRTT + VAR$. Variables T_{first} and T_{last} are the times the first and last acknowledgements are sent during stage i. $SRTT$ is our smoothed round-trip time estimate, and VAR is the estimated variance

We use the RTT estimation technique we discussed in chapter 4 to infer path RTT passively at the wireless client side. Currently we assume that the timestamp option is enabled. In fact, in all the servers in our experiments, the timestamp option was enabled. If it were not, we believe that by using techniques such as those presented in [67] to associate packets with their acknowledgements, we could obtain accurate RTT estimates. We leave this for future work.

The new variance measurement is then the difference between the measured RTT and the $SRTT$. To calculate the $SRTT$ and its variance from each measurement, we employ the same algorithm which is used by many implementations of TCP: the new $SRTT$ is calculated using the formula $7/8 \times SRTT + 1/8 \times RTT$, where RTT is the observed value. (The new variance estimate is computed similarly.)

5.2.4 Transitioning Between States

Figure 5.2 shows the state transition diagram. A connection is in *active* state when it is receiving data within a stage. A connection is in *idle* state if it is finished receiving packets from the previous stage and is waiting for the start of the next stage. In general, this is determined by two conditions: (1) the predicted ending time of the current stage has passed, and (2) no packet has arrived for T_{thresh} ms (set dynamically based on the maximum observed interval between successive acknowledgements) since the last outgoing acknowledgement was sent. The reason for the second condition is that the window could be delayed in the network, and the timer prevents us from transitioning the WNIC to *sleep* mode prematurely. Conditions (1) and (2) combine to serve as safeguards against network delays, which would otherwise lead to prematurely marking

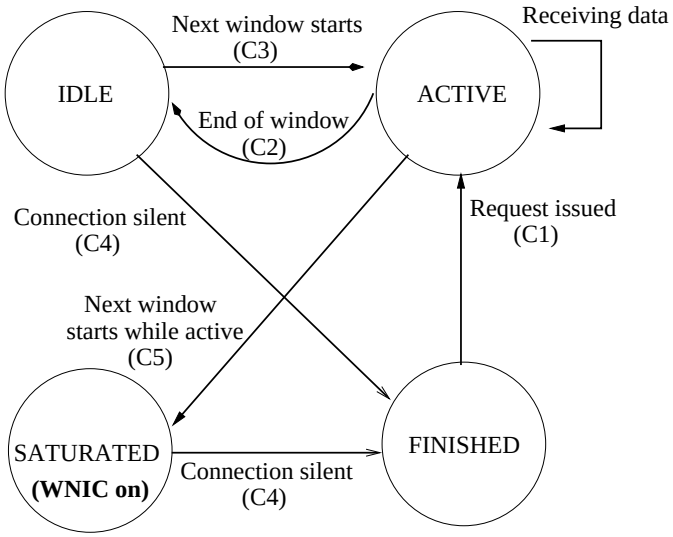

Figure 5.2: State machine demonstrating our algorithm.

a connection as *idle*. When the end of the stage is reached, we store in the connection table the predicted starting and ending times of the next stage.

As HTTP/1.1 uses a persistent connection for successive HTTP requests, connections are not terminated after transmission of a requested object. We mark the connection as *finished* when the requested page has been fully downloaded (and so the client will not receive data on this connection). This is detected using the web page size, which was available over 95% of the time for the web sites in our experiments. If it is not, a connection is never marked *finished*.

Three less common transitions are from *active* to *saturated*, *saturated* to *finished*, and *idle* to *finished*. Transitioning from *active* to *saturated* occurs when sender's window size is larger than the bandwidth-delay product of a connection, and so there is no possibility for energy savings between stages. In this case, we mark a connection as *saturated*.

5.2.5 Extending To Multiple Connections

The technique described above identifies periods when no data is expected on a single connection. Web browsers like Internet Explorer and Netscape generally issue multiple concurrent connections to a site to retrieve embedded objects. The extension to handling multiple concurrent connections on the client is straightforward. In particular, we track each active connection in the client and change its state individually. Each time a connection state changes to either *idle* or *finished*, we check all open connections. The client transitions WNIC to sleep mode only when *all* connections are *idle* or *finished*, and the client keeps the WNIC in *sleep* mode until the *nearest* predicted next stage starting time of all connections.

5.2.6 Round-Trip Time Cache

While the algorithm above is effective in saving energy in many cases, it can be improved. Without prior knowledge, the client cannot predict the `SYN`/`SYN-ACK` and `GET`/`GET-ACK` round trips (the

Site	*Syn-RTT*	*Get-RTT*	*SRTT*
nytimes.com	28	49	33
popupads.com	45	60	49
akamai.com	32	45	38

Figure 5.3: Example Round-Trip Time Cache for a web page request to `nytimes.com`.

`GET/GET-ACK` is often longer than the `SYN/SYN-ACK`). In addition, the round-trip time for the actual data can differ from both. Unfortunately, with multiple connections, the chance that *at least* one connection is in either the `SYN/SYN-ACK`, `GET/GET-ACK`, or first slow start stage can be large. This makes the solution of keeping the WNIC in high-power mode during these phases non-scalable: for highly concurrent connections, little, if any, energy can be saved.

The presence of embedded objects residing on the same server contributes to the high degrees of reference locality user web accesses exhibit high degrees of locality [4]. Therefore, we cache detailed round-trip time information for sites the client has visited within a web page request (see below), including `SYN/SYN-ACK` time, `GET/GET-ACK` time, and the last known *SRTT* (see Figure 5.3). For a given connection C, we start by performing a cache lookup. This information is used to transition the WNIC into low-power mode during the `SYN/SYN-ACK` and `GET/GET-ACK` stages as well as the first slow start stage. If there is no entry for C, we revert to the conservative algorithm.

The key issue is how long entries in the cache should remain valid. Clearly, a cached value stored during off-peak hours is invalid during peak hours and will lead to the client to transition to *sleep* mode for too long, meaning that the `SYN-ACK` packet could be missed. To investigate variance in `SYN/SYN-ACK` and `GET/GET-ACK` times, we downloaded web pages from several sites every half hour, over several days. Investigation of the results show in fact that there is little correlation between `SYN/SYN-ACK` (and `GET/GET-ACK`) times, even when comparing just peak or just off-peak measurements. Hence, our approach to RTT cache consistency is as follows. Between two web page

requests in our trace file, we flush the RTT cache. Within one web page request, we fill the RTT cache with entries for each unique web site accessed. This simple solution has worked well in our experiments (see Section 5.3).

5.2.7 Example

Consider the connection table shown in Figure 5.1, which is a snapshot of the connection table taken at time t=100. When t=102, the connection time for connection 2 expires, ending stage 5 (and marking connection 2 *idle*). At this time, the next stage *end time* can be computed, based on the time the last acknowledgement was sent. If it was sent at time t=95, we predict the *end time* to be 250 (95 + 143 + 12). We next scan the connection table to see if any connections are *active*. No connections are *active*, so we examine the table for the next predicted packet arrival time (based on the minimum *start time* in the table). In this case, the next prediction is for a packet at time t=140, so we transition the WNIC to *sleep* mode until then.

If we next open a new connection to `www.espn.com` at time t=120, we transition the WNIC to *transmit* mode to send the `SYN`, and create a new entry in the connection table (representing a 5th concurrent connection). On this connection, we predict the arrival of the `SYN-ACK` at t=283 using the *Syn-RTT* estimate from the example RTT cache in Figure 5.3, which is 163. We transition the WNIC back to *sleep* mode, until time t=140. Figure 5.4 shows the connection table at time t=130.

5.2.8 Kernel Implementation

We implemented our techniques in a Linux kernel module. It is based on *Netfilter* [80], as we discussed in 3.2. Our implementation filters each incoming and outgoing packet on the client, applying the techniques discussed above. It also maintains the current state of the WNIC. For each incoming packet, we either pass the packet on to the client (if the state of the WNIC is high-power mode) or drop the packet (if the state of the WNIC is *sleep* mode). We patched our kernel using the KURT microsecond resolution timer [55] for our Linux client as discussed in 3.1. Because we

Id	*Status*	*Site*	*Stage*	*Num Packets*	*Next Stage*		*Current Expiration*	*SRTT*	*Var.*
					Start	*End*			
1	*idle*	www.cnn.com	7	—	140	16	—	50	5
2	*idle*	www.espn.com	5	2	232	250	32	143	12
3	*idle*	www.cnn.com	12	—	147	174	—	50	5
4	*finished*	www.cnn-ads.com	—	12	—	—	—	20	5
5	*idle*	www.espn.com	0	—	283	295	—	143	12

Figure 5.4: Updated connection table with a 5th concurrent connection.

use a kernel module, we can run actual Internet experiments as opposed to just simulations. Using this implementation we are able to gain insight into the effectiveness of our solution using real servers.

5.2.9 Limitations

This section discusses two of the limitations of our system. First, in this paper we support HTTP traffic. This type of traffic is two way (request/reply) and predictable. Two-way predictable traffic can be handled as we have in this paper, by transitioning the WNIC between *sleep* and high-power mode when necessary. DNS requests would fall into this category, though because of the likely small round-trip time, the WNIC should probably remain in high-power mode exclusively until the reply. For two-way, unpredictable traffic, the WNIC can be left in high-power mode until the reply is received. Our approach cannot be used for one-way traffic. So, for example, a client cannot use our energy-saving techniques for web downloads while using an application such as voice-over-IP.

Second, our technique successfully saves energy for web sites with a moderate number of concurrent connections. If the number of concurrent connections is large enough, there are not sufficient gaps to save energy, so our algorithm will keep the WNIC in high-power mode for the entire download. However, in such a case, no scheme, including PSM or BSD, is capable of saving

energy, because the connection is saturated. Third, there exist web sites that are not compatible with our technique. One example is any site using *server push* (such as scoreboard pages at `sportsline.com`) to send data at unpredictable times. Note that this case is not a common one.

5.3 Performance

This section describes our experiments and presents our results on web browsing. We first describe how we choose the representative web sites for our experiments. Then, we give our results on real Internet experiments. This includes a comparison with 802.11b power-saving mode (PSM) [20] and BSD [59], as well as a detailed analysis of some of the aspects of our system. It is important to note that we run actual experiments to real Internet servers.

5.3.1 PSM Implementation

We implemented a software version of 802.11b power-saving mode (PSM) [20] so that we could compare it with our client-centered technique (CC). Our implementation of PSM uses a transparent proxy that intercepts packets before they reach the access point and emulates access point PSM behavior. It buffers packets and, every 100 *ms* (the beacon period used by an Orinoco access point), it sends a beacon packet that indicates if any data is buffered. The client responds with an ICMP packet (which emulates the "poll" frame), and then the proxy sends all buffered data to the client. The last packet in a burst is marked so the client can immediately transition the WNIC to *sleep* mode. For BSD, the proxy keeps track of the slowdown parameter (which was either 10%, 20%, 50%, or 100%—this means the slowdown is bounded by no more than that percentage) so that it knows when the client is expecting data.

It is important to note that this implementation of PSM is essentially optimal. We compute the energy postmortem via a client trace and assume no early wakeup whatsoever. Hence, the client is awake waiting for packets only for the time it takes (1) the ICMP packet to travel from the client to the proxy and (2) the first data packet to travel from the proxy to the client. This time is less

than 1 *ms*. In practice, PSM does not work this efficiently; in particular, Chandra and Vahdat found via direct measurement of access points that clients were often kept waiting (in high-power mode) for data after sending the poll frame [13]. This means that PSM is unlikely to perform in practice as well as it performs in our emulation.

5.3.2 Experimental Setup

We use the emulated experimental environment as described in 3.4. We carried out our experiments by running a script that retrieves 100 web pages over a total browsing time of 5,400 seconds. These retrievals generated 558 requests for subobjects, and a total of 2.79 MB was downloaded.

The web pages were chosen as follows. First, we selected the 20 most popular web sites (denoted *top-level sites*) as determined by the Alexa Top Sites web pages [3]. Note that the *reach* is provided by *alexa*, which is the percentage of Internet users that visit that site per day. For each top-level domain, the *alexa* list also provides the probability of visiting each of its subdomains, provided the user stays within that top-level domain. Note that *alexa* does *not* provide the probability that the user stays within the top-level domain. For each site, we include all sub-sites that have a probability larger than 2%.

Our next step is to generate a sequence of web page requests, which is done by using the Alexa probabilities and reach—first we compute the probability of visiting a site given the reach; then, we use the conditional probability of each site in the domain if the next request is in that domain. For think times in between requests, we use the same method for determining think times as used in the BSD work [59] (a Pareto distribution with $\alpha = 1.5$ and $k = 1$). We compare our client-centered technique (CC) with PSM and BSD [59]. For our simulated experiments, we ported an available implementation of these algorithms to ns-2.26 (which has support for HTTP 1.1). Our emulation of PSM and BSD for the actual Internet tests uses a transparent proxy as described in 5.3.1.

In each experiment, a client executes Mozilla, version 1.2 with HTTP 1.1 support (with pipelining disabled), and requests a specific web page. We ran tests both with and without CC. Note that the tests without CC use regular TCP, where the WNIC is maintained in high-power mode and thus does not save energy. In all experiments, we performed each test three times and report the results from the test with the median transfer time. It serves as a baseline to compare against in case our algorithm causes packet loss. The ns-2 simulations use the NSWEB extension [82], which provides support for persistent HTTP connections as specified in HTTP/1.1. The RTT and file size are provided for each server using the values from the real Internet tests (see below).

As described above, we ran our script in both ns-2 and on the Internet. In the former, we use standard ns-2. In the latter, we divide our tests into *peak* tests (run between 12pm and 5pm EDT, which have significant RTT variations) and *off-peak* tests (run between 10pm and 6am EDT). In both, we calculate during execution the amount of time the WNIC is in each of its modes (*idle*, *sleep*, *transmit*, *receive*). Then, we compute energy based on a model of a 2.4Ghz WaveLAN DSSS WNIC, which uses 1319 mJ/s when idle, 1425 mJ/s when receiving, 1675 mJ/s when transmitting, and 177 mJ/s when in sleep mode [42]. Also, we model the energy cost of transitioning the WNIC from *sleep* to *idle* mode as 2 *ms* in *idle* time [59].

5.3.3 Results

Figure 5.5 shows that CC is superior to PSM in both energy consumption and transmission time. It is also superior to BSD in energy consumption. For round trip times less than $100ms$, all BSD variants are equivalent in transmission time (and energy) to baseline TCP.

The figure also shows several CC variants, each with different behavior during think times. BSD is restricted to waking up more often than the CC variants (at least once every 0.9 seconds, to ensure its transmission bound). This means that CC can improve upon the best BSD variant in terms of think time behavior (BSD-100). Figure 5.5 also shows the benefit of using the RTT cache, which is discussed further below.

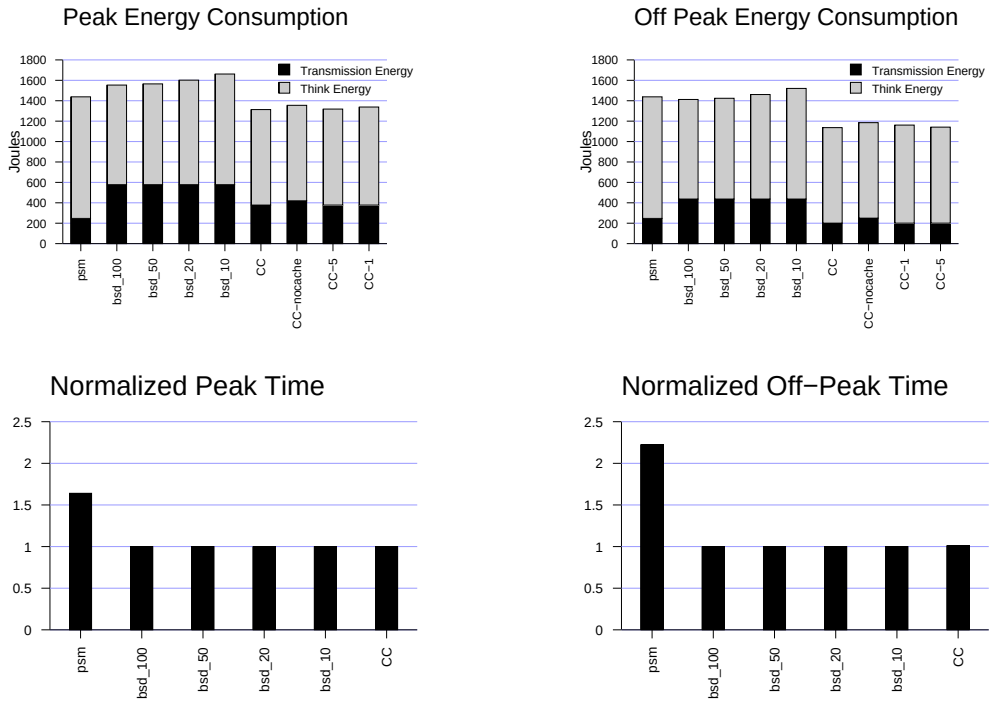

Figure 5.5: Energy consumption and transmission time for both peak and off-peak real Internet tests, normalized to baseline TCP. CC, PSM, and several BSD variants are shown. Smaller bars are better.

We consider successively the median, best, and worst energy savings for each site. In addition, we do the same for transmission time. We calculate the median energy savings by first choosing the test achieving the median energy savings for each site. We then compute the sums over *all* sites of (1) energy consumed and (2) transmission times. The best and worst energy savings are computed similarly. This allows us to examine the expected performance of our technique but also the extremes—the worst case is especially important.

As figure 5.6 shows, our technique results in significant energy savings in the median and best cases. Even in the worst case, there is still some energy savings. The energy savings are more substantial during the off-peak times because there are generally fewer missed packets. The reason for the large increase during off-peak times for the worst case is that for a few sites, the client

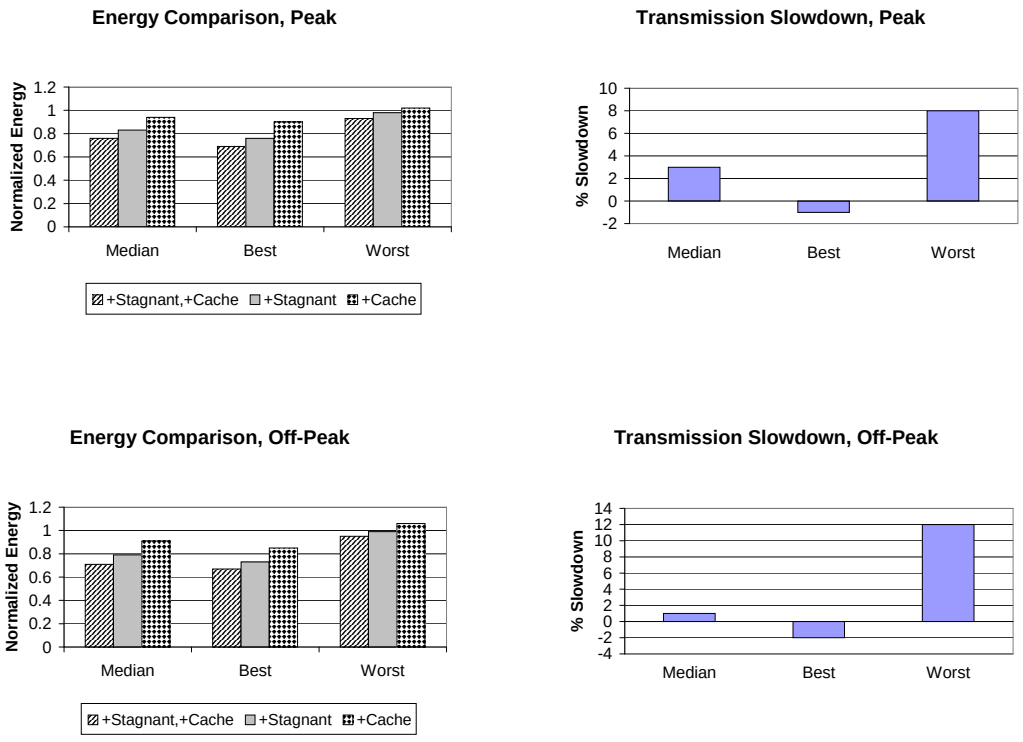

Figure 5.6: Normalized energy and slowdown for both peak and off-peak tests (smaller bars are better). Results shown are the sum over all web sites, taking either the median, best, or worst case for each site.

misses the `SYN-ACK` due to a stale cache entry, causing a large time and energy increase. With a more sophisticated caching algorithm, that both accounts for time of day and time between accesses to the same web site, this problem could be alleviated. We are currently investigating such caching algorithms.

Our technique saves energy for most sites. Figure 5.7 shows the sites for which the least and most (five each) normalized energy is consumed at both peak and off-peak times, as well as the normalized transmission time. In the best case, our technique saves more than 60% of the energy consumed by the WNIC. However, as can be seen, `ebay.com` consumed 83% *more* energy using our technique. In this case, a `SYN-ACK` packet is dropped (by just 1 *ms*) due a slightly faster round-trip time than that stored in the RTT cache. This resulted in a client timeout, all of which is spent in

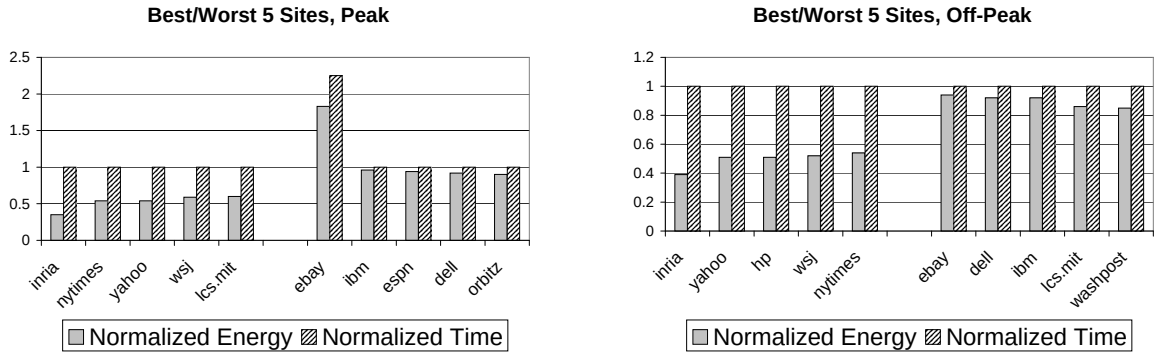

Figure 5.7: Normalized energy consumed and transmission time for best and worst five sites during both peak and off-peak tests (smaller bars are better).

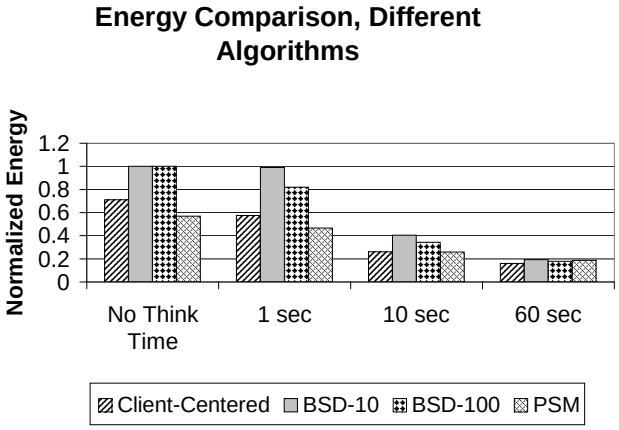

Figure 5.8: Normalized energy for our technique (client-centered) compared to BSD and PSM (smaller bars are better). We experimented with four different think times in between each web request. The average transmission slowdown for PSM is 40%, whereas with our technique it is 4%.

high-power mode. Such misses can occur, for example, because of DNS redirection—this means the site cached is not the same as the one we are accessing.

Figure 5.8 shows (off-peak) results of energy savings using our client-centered technique, PSM, and BSD-10 and BSD-100. BSD-10 means that the slowdown parameter is 10%, and BSD-100 means that it is 100%. We used four different "think times" between each web site request: 0, 1, 10, and 60 seconds. This experiment shows that our technique consumes less energy than BSD-10 and BSD-100. Comparing our technique to PSM shows that PSM consumes less energy for smaller think times, as is expected. However, the expense of beacon packets causes PSM to consume more energy than our technique for the 10 and 60 second cases. Finally, the BSD algorithms are better than PSM only during think times, so for 60 seconds (and larger), BSD saves more energy than PSM.

We also computed the average transmission slowdown for all active user periods, although only for our technique and PSM. Clearly, PSM has a much larger slowdown than our technique, 40% to our 4%, because with PSM all RTTs are rounded up to the nearest 100 *ms*. The BSD tests have the same transmission time as regular TCP. This is because under the BSD algorithm, the WNIC remains in high-power mode for all periods of user activity. Specifically, the BSD-10 and BSD-100 algorithms must leave the WNIC in high-power mode for 1 second and 100 *ms*, respectively, after each outgoing packet, including acknowledgements. In all but one of our tests (`inria.fr`), RTTs were less than 100 *ms*.

5.3.4 Detailed Analysis

This section gives an in-depth analysis of several aspects of our system. We first break down the different components of energy. Second, we show the improvement gained from using the RTT cache. Finally, we investigate the effect of different RTTs on energy consumption.

Figure 5.9 quantifies the sources of energy consumed in our experiments. Energy can be divided into five categories: *early wakeup*, *late sleep*, *inter arrival*, *receive*, and *sleep*. Early wakeup energy accounts for energy consumed between the transition to *idle* mode and the receipt of the next

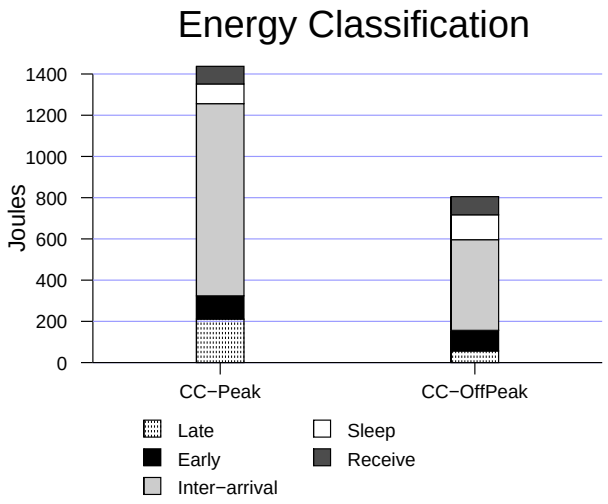

Figure 5.9: Breakdown of energy consumed for CC, in joules, for both peak and off-peak browsing sessions.

packet. Late sleep energy accounts for the time spent between the transmission of the last acknowledgement and the transition to *sleep* mode. Inter-arrival energy accounts for the time spent in *idle* mode while actively receiving packets. Receive energy is the energy consumed when actually receiving data. Sleep energy is the energy consumed when the WNIC is in *sleep* mode. Note that any energy-saving technique will contain some of each of these categories (other than optimal, which has no *early wakeup* or *late sleep*).

Figure 5.10 shows that the RTT cache is effective in saving energy during downloads. In particular, the overall benefit (far right) is about 10% (compared to the regular TCP test). The rest of the figure breaks down the benefit of the RTT cache for each number of concurrent connections. In particular, the largest benefit of the RTT cache is at 3 and 4 concurrent connections, providing an improvement of 13% and 32%, respectively. With 1 and 2 concurrent connections, the benefit is limited because the RTT cache pays off only during connection setup. With 5 and 6, there is a high likelihood that connection setup overlaps with an active stage for a different connection.

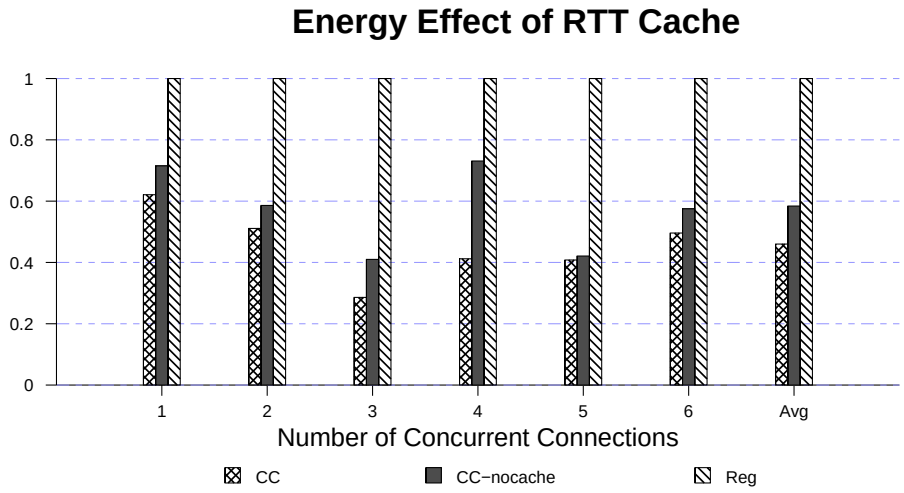

Figure 5.10: Benefit of RTT cache. All results are normalized to regular TCP, during downloads only.

Figure 5.11 shows energy savings for the five smallest and five largest round-trip times. Theoretically, lower round trip times have less potential to benefit from CC, because there is less time to sleep between stages. Also, connections with low round-trip times are more likely to be placed in *saturated* state, during which the WNIC does not transition to *sleep* mode. As an example, `inria.fr` has an RTT of 133 *ms*, and we save over 50% when downloading from that site. On the other hand, `cnn.com` has a 7 *ms* RTT, and it saves only 28%. Still, the energy saved when accessing `cnn.com` is surprisingly large. The reason we are able to save energy with such a short RTT is because the client is able to declare all connections stagnant—this allows the WNIC to be transitioned to *sleep* mode, whereas the naive client simply waits for further data. Finally, `espn.com` achieves the least energy savings despite its large round-trip time. This is because it has between 9 and 11 concurrent connections, so potential for energy savings is limited.

Figure 5.12 compares CC to the optimal energy consumed for the best and worst five sites during off-peak hours. Optimal energy is what one could do if an oracle predicted perfectly when to transition the WNIC into high- and low-power modes; i.e., the WNIC is in *sleep* mode at all times

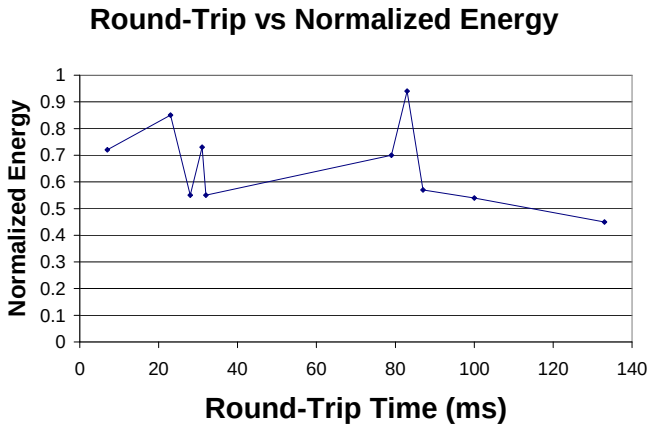

Figure 5.11: Round-trip time vs. normalized energy consumed for the five smallest and five largest round-trip times.

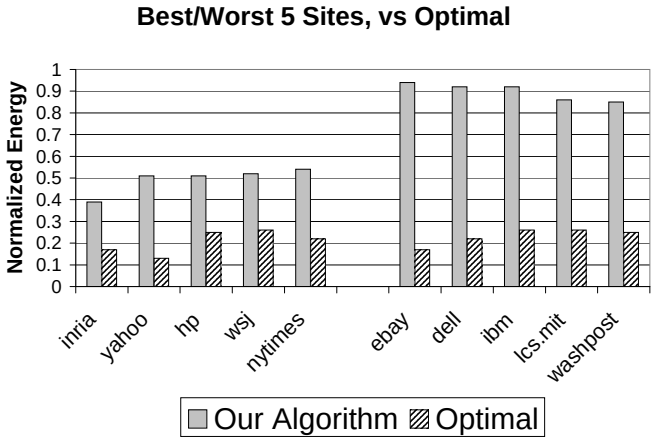

Figure 5.12: Comparison of CC to optimal, during off-peak hours, for the best and worst five sites.

except to receive packets. For our five best tests, CC saves between 45 and 60% energy, compared to 75 and 90% using the optimal algorithm. We believe this is quite good. The five worst tests are not as good—CC saves on average about 12% energy, where optimal saves over 70%.

5.4 Summary

In this chapter, we have discussed our client-centered technique for energy efficient communication during web browsings. We examined the characteristics of web traffic, identified opportunities for energy efficient web browsing, and proposed a set of client-centered technique to exploit them. Our algorithm does not require changes to TCP or web servers; furthermore, it requires no proxies or use of 802.11b power-saving mode. Our algorithm maintains the state of each client initiated connection and transitions the WNIC to a lower-power *sleep* mode when all connections are idle. It then transitions the WNIC to high-power mode before the next packet arrives. The key implementation mechanisms are (1) a connection table that tracks the state of each connection and (2) an RTT cache that allows the WNIC to be transitioned to *sleep* mode during connection setup.

Results show across all experimented web sites, the median energy savings is over 20% and the median increase in transmission time is less than 5%. In addition, our client-centered technique is better than PSM in terms of transmission time and better than BSD in terms of energy savings. We feel that our energy-saving techniques will help prolong battery life on mobile devices without significantly affecting the user web browsing experience.

In the next chapter, we will describe another set of techniques, this time for large TCP downloads.

Chapter 6

Energy Efficient Large TCP Downloads

In the previous chapter, we discussed and evaluated a set of client-centered technique for energy efficient communications for web browsing. As discussed in Chapter 1, the Internet and wireless networks are experiencing dynamic and exciting changes with new services and devices. Work in [97] points out that the Internet has experienced an astronomical increase in the use of content delivery networks and peer-to-peer file sharing systems. As a result, TCP downloads are rapidly becoming another important source of Internet traffic. In this chapter, we examine the trade-offs between energy and performance for large TCP downloads.

First, we describe the energy problems associated with large TCP downloads in a wireless environment. Then we present implementation details and explain concerns related to our techniques. We evaluate our technique with extensive experiments and give a thorough discussion of the performance results.

6.1 Large TCP Downloads

Content delivery is becoming pervasive and more important in recent years, especially with the surging popularity of peer-to-peer file sharing systems. Generally, file sharing depends on TCP downloads because TCP provides reliable data delivery service between the communication peers. From the "old-fashioned" FTP downloads to today's Kazaa [56] or Gnutella file peering [36], a long time TCP session is normally required for exchanging large amounts of data.

When large files are downloaded through TCP connections, the client saves the most energy when data is transmitted in bursts, enabling it to transition its WNIC to *sleep* state between each

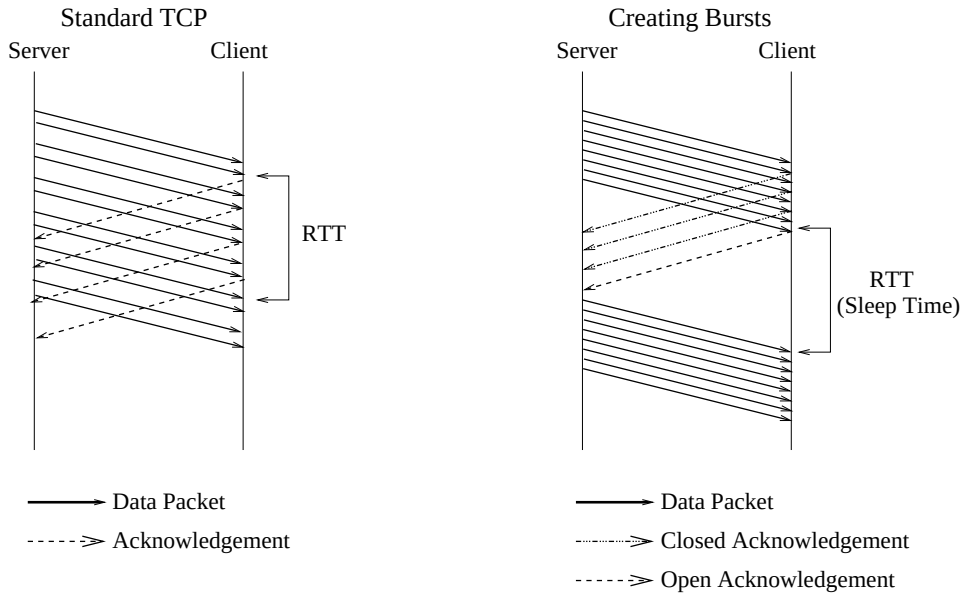

Figure 6.1: Example of creating bursts. On the left is standard TCP, where the packet stream is smoothed, and on the right is our client-centered technique. To create bursts, the first three acknowledgements advertises a receiver window size of 0, and the fourth advertises a full buffer. This creates more potential time the WNIC can remain in *sleep* mode, though the transmission will increase.

burst. However, TCP does nothing to combine packets into bursts—instead, it attempts to smooth the packet stream—which makes saving energy during long transfers difficult due to significant consumption when the WNIC is in *idle* mode. When bursts do appear naturally in a TCP stream (e.g., during slow start or due to ack compression), they are unpredictable and so not amenable to energy savings.

We have designed and implemented a technique for TCP downloads in mobile clients that is able to save energy during the data transmissions. The basic idea to shape the TCP traffic into bursts at agreed-upon intervals and transition WNIC to a lower-power *sleep* mode during those intervals to save energy. In principle, this technique trades reduced WNIC energy for increased transmission time.

6.2 Energy Efficient Techniques

In this section, we describe the design and implementation of our client-centered techniques for the large file downloads. As discussed in Chapter 2, TCP uses congestion control and flow control to limit its data sending rate without overloading the network and the receiver. This rate limitation is achieved by a sliding window scheme that controls the number of in flight packets (sent but not yet acknowledged) over each round-trip time. The congestion window size in TCP is dynamically adjusted according to network conditions. For example, the window size shrinks when congestion occurs, while it is usually increased if all packets in a window are acknowledged. In addition to the window-based congestion control, TCP uses a *self-clocking* mechanism to pace outgoing packets within a window. Instead of sending all the packets in a window at once, TCP only allows sending a new packet when another packet is acknowledged. This results in a smoothed data stream when the acknowledgments are evenly paced.

Our implementation exploits TCP for energy savings. By controlling the acks fed back from the wireless client to the sender, the receiver implicitly forces the sender to send each window of packets in a burst so that potential sleep time between bursts is maximized. In the following, we first discusses the basic idea behind our client-centered technique (CC) and give low-level implementation details. This implementation works with client-side modifications to TCP, and so it can be more easily deployed.

6.2.1 Client-Centered Traffic Shaping

Our goal is to shape a smoothed TCP data stream into bursts to increase potential sleep intervals between packet receptions. To do this, we exploit TCP's flow control mechanism at the receiver (client) to manipulate the sender (server). In TCP, each acknowledgement from client to server contains the client's advertised receiver window size, which is the number of new packets it is able to hold in its buffer. In our modified TCP implementation, the client first announces zero buffer space to the sender to delay outgoing data packets at the server, and later announces appropriate buffer space to allow the server to release packets in a burst. Specifically, in each but the *last*

acknowledgement in a window, the client advertises its receiver window size as zero (denoted as a *closed ack*). When the server receives a *closed ack*, it cannot send any further packets, as it believes that the receiver has no available buffer space to store them. When the receiver believes the window has completed, it triggers the next window of packets by sending an acknowledgement with the window size advertised by TCP (usually 64KB). We denote this kind of acknowledgement as an *open ack*. Because the *open ack* (implicitly) acknowledges the (entire) previous window, the server will immediately send the (entire) next window. Figure 6.1 shows the difference between standard TCP and our modified TCP.

Our client-centered technique converts a smooth TCP stream to a bursty TCP stream. This creates large gaps during which is possible to transition the WNIC to *sleep* mode. In order to decide when to transition into and out of *sleep* mode, the client must infer two things: when a burst of packets ends, and when the first packet of the next burst arrives.

6.2.2 End of Burst Detection

The detection of the end of a burst is nontrivial because of the variance in round-trip times inherent to the Internet. The basic idea is for the client to infer the end of a burst. when no packets arrive for a threshold amount of time.

Clearly, there is a tradeoff between inferring the end of a burst aggressively (i.e., using a smaller threshold) and conservatively (using a larger threshold). The former increases the risk of missing packets, because the packets could merely be delayed (due to round-trip time variation) and arrive shortly after the client transitions the WNIC to *sleep* mode. Such a situation directly leads to missed packets and can potentially cause an *increase* in energy consumption (as well as unacceptably long transmission times). The latter always increases both energy and transmission time overhead, as there is extra time spent attempting to correctly infer the end of the window, and it is spent in *idle* mode.

Three broad approaches to end-of-burst detection are possible. We briefly discuss the first two, which passively predict the end of a burst through a (1) a fixed threshold and (2) a dynamic threshold. Then we discuss in detail a novel technique we call *active burst detection*, which can in most cases precisely determine the end of a burst.

Note that both the dynamic threshold and *active* assume that the client can maintain an accurate estimate of window size (W) in each round. Our current algorithm to estimate W sets the new window prediction as one more than the number of packets seen in the last window, to mimic what TCP does in its steady state.

Passive Burst Detection Our approach to this problem is to implement both fixed and dynamic thresholds and compare their performance. One general idea for determining the end of a burst is what we call *passive burst detection*, where the client infers the end of the burst passively. The basic idea is for the client to infer the end of a burst when no packets arrive for a threshold amount of time. The first possibility for this is to use a fixed threshold, which is straightforward: if no packet is received for T_f ms, we conclude that the current burst has ended, and the client sends an *open ack* to the server. The advantage of such an approach is that it is simple to implement and has minimal overhead. The disadvantage is that it cannot adapt to different network conditions. In particular, when network transfers have low variance (e.g., late at night), it may be excessively conservative. When conditions are poor (e.g., 3pm on a weekday), a fixed threshold may be too aggressive, transitioning the WNIC to *sleep* mode before a burst is complete. Similarly, it could mistakenly transition to *sleep* mode because of the time gap caused by packet loss.

We also studied a dynamic approach. Because each window is sent in a burst, we can use the interarrival times between packets and their variance to infer the end of a burst. We use each interarrival time as a sample and then use a scheme that is analogous to what TCP does when computing the timeout value. That is, given new samples for arrival (S_A) and deviation (S_D), we

keep a running average of interarrival times (I_A) as well as interarrival deviation (I_D):

$$I_A = \frac{7I_A}{8} + \frac{S_A}{8}$$
$$I_D = \frac{3I_D}{4} + \frac{S_D}{4}$$

Given these estimates, we use the following intuition for detecting the end of a burst. The more packets that are received, the more likely the burst is over. Assuming that the client knows the sender's congestion window size W (this inference is discussed below) and that the client has received $P < W$ packets, then any of the rest of the (expected) $W - P$ packets in the burst could arrive in $T = (W - P)I_A$ time. To allow for variance, we increase T by $4I_D$, as used in the TCP RTO estimation. Specifically, the dynamic threshold (T_d), which is a function of P, for determining the end of the burst is:

$$T_d(P) = (W - P)I_A + 4I_D$$

This means that the threshold value decreases as more packets come in during a burst.

The above algorithm assumes that we have an accurate estimate of window size (W) in each round. Our current algorithm to estimate W sets the new window prediction as one more than the number of packets seen in the last window, to mimic what TCP does in its steady state[1].

One problem with our samples of I_A and I_D is that if not all packets arrive (due to packet loss), the estimate of I_A and I_D may not be correct. Our current approach in this case is to double I_D and increase I_A by that amount. Furthermore, in practice the upper bound of T_d is set to the one-way trip time, and the lower bound is set to 3 ms, which in our experience is the minimum reliable resolution for our timer.

Active Burst Detection While a dynamic threshold is typically a better idea than a fixed one, both will at times suffer from being either too conservative or too aggressive. As discussed above, this will lead to wasted energy, missed packets, or both. This is especially true when variation in

[1]Slow start and packet losses are handled separately in a different state; the states and their transitions are described in Section 6.2.6.

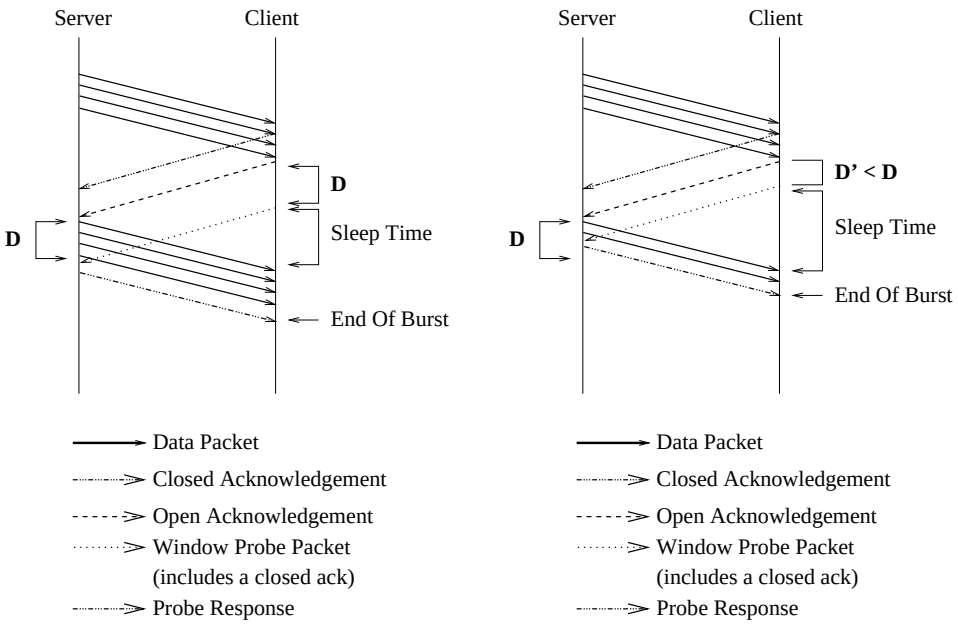

Figure 6.2: Example of *active*. The left-hand side shows correct operation. The right-hand side shows a case where the window probe packet arrives in the middle of a burst, preventing part of the window from being sent by the server.

round-trip times is significant, which can occur due to network variation or multiple clients sharing an access point.

We have developed a novel technique for precise end-of-window prediction that we call *active burst detection*. The basic idea is for the client to convince the server to inform it when a burst is over, i.e., the client actively determines the end of the burst. If this can be done, the client can transition the WNIC to *sleep* mode immediately upon this determination. Active burst detection allows in practice for the client to make near-optimal end-of-burst predictions.

The basic idea is as follows (see the left-hand side of Figure 6.2). When an *open ack* is sent to the server, the client delays for time D (see below). Then, the client sends a carefully crafted *window probe packet* to the server. In practice, this packet is essentially a zero-length, out-of-order packet. A TCP server responds to such a packet by sending an acknowledgement, which we call

the probe response. As long as D is sufficiently large so that the probe response arrives after the last packet in the burst, the client can safely declare the burst over and transition the WNIC to *sleep* mode.

Three main issues arise with the use of a window probe packet. First, the server will respond immediately to the window probe packet, even if the entire burst has not been released due to server delay. This would cause the window probe packet to arrive before the end of the burst, leading to an incorrect transition of the WNIC to *sleep* mode. To prevent this problem, we send a *closed ack* immediately after the window probe packet. This way, if there is a server delay, the remaining part of the burst will not be sent until the next *open ack* is received. A related problem is that the window probe packet can be "piggy-backed" on the last data packet if the time between the *open ack* and window probe packet is not large enough.

Second, the choice of D is a tradeoff between (1) decreased energy savings and increased time overhead and (2) a larger window size and accurate end-of-burst detection. In other words, the larger D is, the longer the client waits in high-power mode before sending the *open ack* and the window probe packet. However, the smaller D is, the more likely it is that either (1) the window probe packet sent by the client arrives at the server too early (shown in the right-hand side of Figure 6.2), which prevents the sender from sending all of the packets in a window, or (2) the probe response sent from server to client is piggybacked or reordered with at least one data packet somewhere on the network, leading to an incorrect end-of-burst detection.

Our current algorithm for determining D is to initialize $D = W/B$. In this formula, W is the client's estimate of the sender's congestion window size and B is the estimate of server bandwidth. The idea is that W/B approximates how long it takes the server to release an entire burst. Ideally, if we know the server bandwidth and that there are no variations in round-trip times, we can simply use this D for the lifetime of the download.

However, in practice neither of these is the case, so we allow D to adapt. To calculate the initial D, we currently set B to 100 Mb/s (because of the popularity of Fast Ethernet). Variable D is increased by 1 *ms* (the smallest amount the timer can support) in two cases: (1) whenever there is reordering of the probe response that causes it to arrive in the middle of a burst, or (2) whenever the probe response is piggy-backed on the last data packet of a burst. We can detect the former situation because the WNIC is not transitioned to *sleep* mode until the window probe packet is sent, which occurs D *ms* after receiving the probe response. We can detect the latter case through the TCP timestamp or a sub-MSS sized packet.

We decrease D when we have seen P bursts in a row without either of the two cases above occurring (currently P is set to 10). The decrease in D is half the average gap between the last packet in a burst and the window probe packet. We also provide a floor and ceiling for D; currently this is 3 *ms* and the one-way trip time (1/2 RTT). If it would be necessary to increase D above the ceiling, we instead switch to the dynamic threshold algorithm, as this indicates that the server is heavily loaded and cannot promptly send packets.

Third, the window probe packet could be lost. When this occurs, the client keeps WNIC in high-power mode for at most a one-way trip time. If a loss occurs at the largest value of D that we consider, we switch to passive detection with a dynamic threshold. In practice, this is rare because the small size of the window probe packet and low reverse data path loss (see below).

6.2.3 Next Window Prediction

Once the end of the window is detected, an *open ack* can be sent and the WNIC transitioned to *sleep* mode. To avoid missing packets, the WNIC needs to be transitioned back to *idle* mode before the first packet of the next burst. That packet will arrive approximately RTT *ms* after the *open ack* is sent. This means that we must keep an accurate estimate of the RTT on the client. We use the technique in chapter 4 to estimate the round-trip time of a connection.

We added measurements at the TCP receiver to produce RTT samples for every packet. Given accurate RTT estimates, we use the the minimum RTT over all samples. This is a conservative approach, resulting in some wasted energy but also almost always avoiding missed packets on the client. In practice, this scheme performed quite well (see Section 6.3).

6.2.4 Challenges

In this section, we discuss two challenges that arise when implementing CC: saturated connections and lost *open acks*.

Saturated Connection Energy saving is only possible when the bandwidth of the wireless network is not fully used. This is because there must be time that the WNIC can be transitioned to *sleep* mode. During downloads from some Internet sites, the wireless network is saturated. If the client executes the energy-saving algorithm described above, the result will be longer transmission time and *increased* energy usage. Note that when the wireless network is saturated, *no* technique can save energy.

To handle this, the client uses its estimate of window size, wireless bandwidth, and the round-trip time to determine when executing the energy-saving algorithm is not profitable. If the window size is within a threshold (currently 90%) of the bandwidth-delay product (wireless bandwidth multiplied by the round trip time) for three consecutive bursts, we revert to standard TCP. We do not attempt to resume saving energy if the bandwidth-delay product decreases, because our experience to date is that a saturated connection almost always remains saturated.

Lost *open acks* One problem with CC is that TCP is now vulnerable to the loss of *open ack* packets. Whenever an *open ack* is lost, if the client takes no action, the server will time out and probe the client. This is because in the absence of reception of an *open ack*, the server believes the client has no buffer space to store packets—it has previously received a series of only *closed acks* during the current burst. A timeout causes a large overhead in both energy and time, because (1)

the client spends significant time waiting for packets in *idle* mode, and (2) the sender cuts the congestion window size to one packet.

Our current approach to this problem is to have the client wait twice the estimated RTT after the original *open ack*. If no data has arrived, it then retransmits the *open ack*. An *open ack* is retransmitted each RTT *ms* if the next burst does not arrive. In practice, this technique was sufficient in our experiments, because even under network conditions during peak times, loss of an *open ack* was rare. In particular, our experiments showed that no Internet site we used in our test suite incurred more than one lost *open ack*. This is not surprising, given prior research that indicates that more than 90% of loss is on the data path [17].

6.2.5 Excessive Acknowledgement Delay

To increase the chance of piggybacking acknowledgments, most TCP implementations by default use delayed acknowledgment. This means that other than during slow start, TCP receivers usually send acknowledgements every other packet. This is problematic if an odd-sized window occurs. For example, assume that an odd-numbered packet in a given window is received but no even-numbered packet arrives. The TCP receiver delays for some amount of time (40 *ms* in Linux 2.4, for example) before assuming that the even-numbered packet will not arrive. At that point the receiver sends an acknowledgement.

This TCP behavior is not energy efficient, because the time spent waiting for an even-numbered packet is spent in *idle* mode, which wastes energy. Hence, we use what we call *eager acknowledgement*, which means that the *open ack* is sent based on when the client detects the end of a burst (see above) rather than when TCP itself would send the acknowledgement.

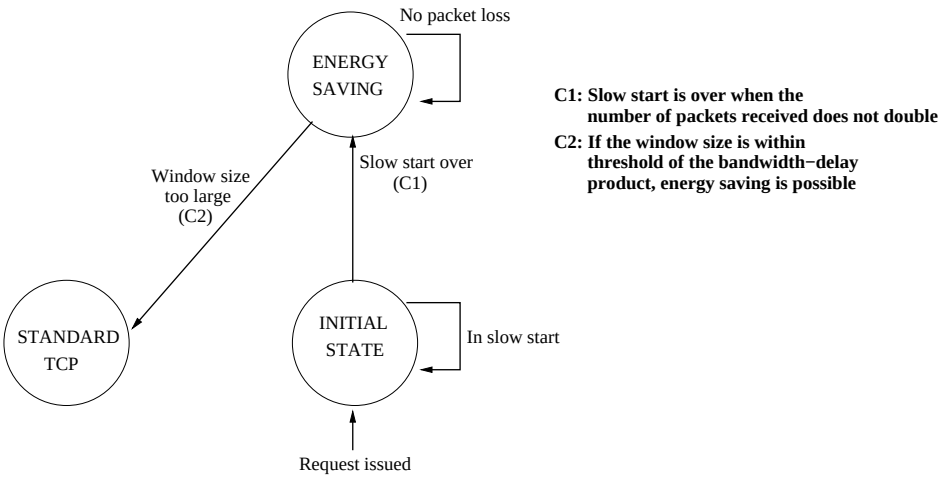

Figure 6.3: States in our algorithm.

6.2.6 Overall Algorithm

Our overall algorithm is shown pictorially in Figure 6.3. The WNIC is kept in high-power mode in all but the *energy-saving* state, in which our client-centered technique is being applied. From our initial state, we transition to our energy-saving algorithm when slow start is over, which the client detects when the number of packets received does not double. We move to recovery mode if a burst is received out of order, and move back to saving energy as soon as a window is received in order. In two cases, we revert to standard TCP: (1) the connection is saturated, or (2) an in-order window is not seen for three consecutive windows.

Effect of Bursts

Our client-centered technique has the potential to affect the queuing behavior of routers, since it purposely introduces burstiness into the packet stream. However, we do not expect the effect to be a significant problem. This is for two reasons. First, individual packet streams already experience some amount of burstiness due to slow start, missing acknowledgements, and blocking at the application layer [41]. In fact, any wireless device utilizing IEEE 802.11b power-saving mode (PSM)

[20] will introduce burstiness into the network if the connection is not saturated—independent of our algorithm—because PSM results in ack compression at the client. Our approach, while incurring slightly more burstiness than PSM, differs primarily in that it attempts to exploit the burstiness, and hence controls it rather than allows it to happen in an arbitrary way. In any case, as effective wireless bandwidths are generally much less than 50 Mb/s, the burstiness caused by a single wireless client should not be too severe.

Second, because CC increases transmission time, the amount of data per unit time that passes through routers *decreases*, thereby having a net positive effect on queue length. We studied the interaction of our energy-saving TCP flows and standard TCP flows using an ns2 [81] simulation. Somewhat counterintuitively, we found that replacing standard TCP flows with energy-saving TCP flows *reduces* packet loss at the routers and increases the throughput of standard TCP flows. The results of our simulation are shown in detail in [120].

6.2.7 Limitations

This section discusses our limitations. First, in this paper our techniques only support client initiated request/response type of TCP streams. This type of traffic is two way and predictable. Although there are potentially many other kinds of traffic that can arrive at a wireless client without client soliciting, we believe this type of traffic is the largest wireless traffic.

Two-way predictable traffic can be handled as we have in this paper, by transitioning the WNIC between *sleep* and high-power mode when necessary. DNS requests would fall into this category, though because of the likely small round-trip time, the WNIC should probably remain in high-power mode exclusively until the reply. For two-way, unpredictable traffic, the WNIC can be left in high-power mode until the reply is received. Our approach cannot be used for one-way traffic. For example, `ARP` traffic as well as voice-over-IP are one way and unpredictable. For many of these kinds of traffic, if a packet is missed because the WNIC is in *sleep* mode, it is not critical. For example, a client can temporarily ignore `ARP` packets (e.g., from peers on the wireless network).

Clearly, though, a client cannot use our energy-saving techniques for downloads while using an application such as voice-over-IP. However, we believe our approach handles the most common cases.

Second, our implementation uses KURT millisecond timers and extra computation to maintain state. This will increase the energy consumed compared to using baseline TCP. While a complete analysis would include all of this energy, the effect is likely small and is certainly difficult to quantify. Hence, we have not included these effects in our performance measurements. (Note that newer Linux versions are moving toward making millisecond timers standard.) Third, we assume that the server is not mobile and so its energy savings are irrelevant.

Finally, we do not consider the effect of the mobile client moving between access points. Clearly, if such a scenario were considered, the WNIC would need to remain in high-power mode during periods of roaming.

6.3 Performance

This section describes our experiments and presents our results. Section 5.3.2 describes our experimental methodology. Section 6.3.2 presents our overall experimental results on 7 Internet servers, including a comparison to traditional 802.11b power saving mode (PSM). Finally, Section 6.3.3 provides a detailed analysis of situations where round-trip times vary as well as the effect of different access point bandwidths.

6.3.1 Experimental Methodology

In our experiments we examined the performance of our system on both real Internet traffic using actual servers and emulated traffic using *DummyNet*. We use the emulated experimental environment as described in 3.4. we used *DummyNet* [95] to experiment with different wireless bandwidths, RTTs, and loss rates. In our experiments we do not differentiate between wireless and wired loss. We show experiments up to 1% loss. Most of our tests used a 20Mb/s bandwidth

Site	*Base RTT* (*ms*)	Window (KB)
`cs.uiuc.edu`	21	61
`hp.com`	58	31
`cs.washington.edu`	60	32
`cs.stanford.edu`	65	61
`jriver.net`	63	11
`research.msft.com`	84	17
`inria.fr`	114	61

Figure 6.4: Sites used in Internet tests with base RTT (without variations) and average window size.

between access point and client, while a few use other bandwidths. The 20 Mb/s value was selected by experimenting with a 54 Mb/s access point and measuring the peak bandwidth attained on a wireless client. In practice, most wireless TCP connections cannot achieve 20Mb/s, because of the way that default socket buffer size is chosen in most systems (e.g., for a 64KB TCP socket buffer at a 40 *ms* RTT, the maximum speed is below 20 Mb/s). Hence, the wireless link is not the bottleneck, which allows energy saving.

Experiments In each experiment the client performs an `ftp` download from the server, requesting a file between 4MB and 5 MB. For each experiment, we ran both our algorithm and a baseline test. The baseline test (denoted "baseline TCP") used the standard TCP implementation in the Linux kernel. The baseline TCP version keeps the WNIC in high-power mode for the duration of the download. We compute normalized energy savings and transfer times by comparing to the baseline. In all experiments, we performed each test at least 10 times and report the results from the test with the median transfer time.

Internet Experiments The Internet experiments were carried out by running a script that downloaded a file at a time, in succession, from seven Internet servers shown in Figure 6.4. The client

machine accessed the Internet via the emulated access point. Experiments were performed during peak traffic times (2-5PM EST) as well as off-peak traffic times (10PM-1AM EST).

DummyNet Experiments In order to study our system in an environment where experiments are mostly repeatable, we used *DummyNet* to construct a emulated environment in which we could control loss rates and round trip times. We model both peak and off-peak traffic patterns of the Internet. We observed that *DummyNet* is able to emulate well the lack of significant round trip variation that occurs during off-peak traffic times. Hence, our off-peak simulations used the standard *DummyNet*, with a variety of round trip times (30 *ms*, 60 *ms*, 90 *ms*, and 120 *ms*) and loss rates (several between 0% and 1%). However, peak traffic times show a much higher degree of variation in round trip times. Unfortunately, *DummyNet* does not handle round trip time variation; it uses only fixed delay values for a particular path. To address this, we modified *DummyNet* to add RTT variation without causing out-of-order packet arrivals, as was done in [72]. We model round trip time variation using an uncorrelated gamma distribution [78]. We obtained this distribution by performing a ping test during peak time, gathering several thousand samples from the `ftp.cs.washington.edu` server (which has a 61 *ms* base RTT), and then using these samples to determine the parameters to the gamma distribution. Note that while a realistic distribution is likely correlated, (1) *DummyNet* itself cannot handle a correlated distribution, and (2) an uncorrelated distribution is *more* difficult for CC to handle effectively than a correlated one, due to the increased unpredictability of round-trip times. To simulate peak traffic on round-trip times other than 60 *ms*, we scaled this gamma distribution proportionally to the new round-trip time.

6.3.2 Overall Results

In this section we discuss the performance of CC in two environments. CC uses *active* unless otherwise noted. All results are normalized to baseline TCP.

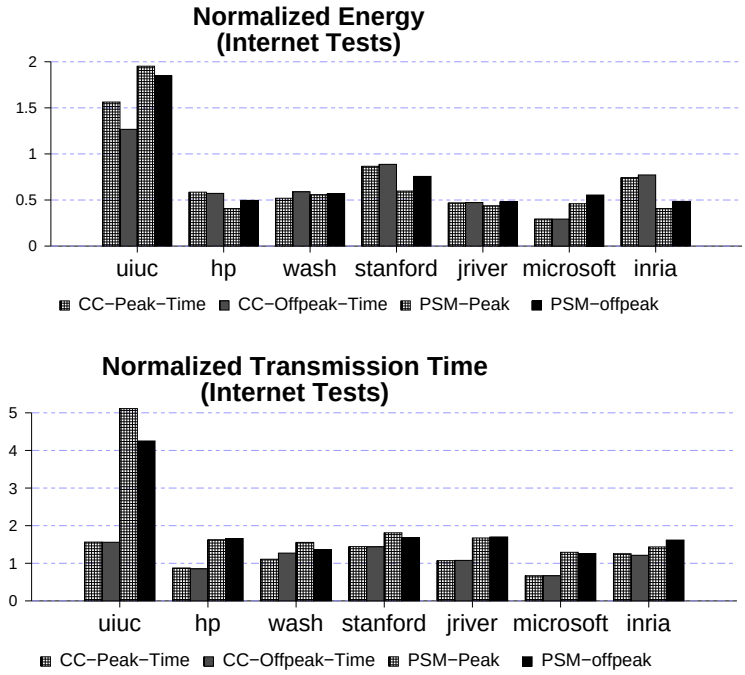

Figure 6.5: Overall normalized energy savings and transfer time results for Internet tests using a 20 Mb/s access point and active end of burst detection during peak (2-5PM EST) and off-peak (10PM-1AM EST) times. Results are shown for both CC and PSM.

First, Figure 6.5 shows energy consumed and transmission times for both peak and off-peak times when downloading files from several Internet servers. Second, Figure 6.6 shows the same metrics using emulated traffic with *DummyNet*. In both environments, our system is effective at saving energy while maintaining reasonable download speeds. During peak times on the real Internet, our system on average uses 27% less energy with a increased normalized in transfer time of 20%. During off-peak times, our energy savings is 32% with a increased normalized transfer time of 20%. In our simulated peak environment our average energy savings (over all round-trip times and loss rates) is 51%, with an increased normalized transfer time of 12%. In the simulated off-peak experiments, energy savings is 47% and the normalized transfer time increases by 6%.

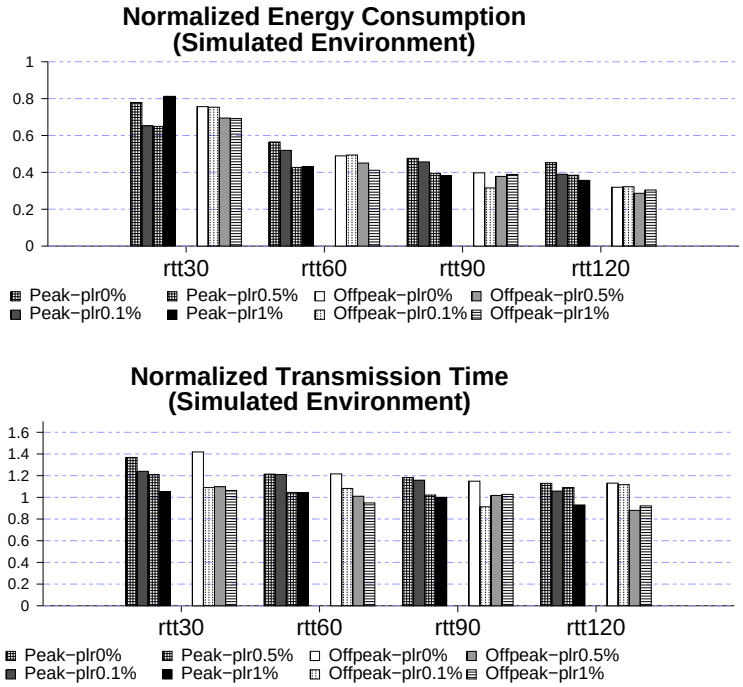

Figure 6.6: Overall normalized energy savings and transfer time results for *DummyNet* tests using a 20 Mb/s access point and active end of burst detection during emulated peak and off-peak times. Here, only CC results are shown.

Internet experiments The primary time overhead of CC is stream delay caused by *closed acks*. (The overhead D is typically much less than that of *closed acks*.) Therefore, the performance of CC is dependent on two factors: the round-trip time and the average window size. This is because the *closed ack* overhead decreases as (1) round-trip times increase and (2) average window size decreases. This can be seen from Figure 6.5, in which sites are shown ordered by increasing RTT. The time overhead does not strictly decrease as RTT increases, because the average window size is not the same for all sites (see Figure 6.4). A typical value for the window size is about 32 KB. The energy savings for all sites other than `cs.uiuc.edu` is at least 10%, and is often around or more than 50%.

There are a few sites for which the average window size varies significantly from 32 KB, which impacts performance considerably. One is with our best case, `jriver.net`, where the energy savings is more than 50% with less than 8% increase in transmission time. This is because this particular site has a small average window size (11 KB). On the other hand, both `cs.uiuc.edu` and `cs.stanford.edu` have a window size of nearly the maximum (64KB). Combined with the low RTT for `cs.uiuc.edu`, this causes CC to revert to baseline TCP. The reason for the large overhead is (1) startup overhead, when our client is trying to synchronize into energy-saving mode, and (2) energy-saving overhead, which occurs in the three round trips before reverting. The entire 5MB file takes only 2 seconds to download, so these overheads are not well amortized. For `cs.stanford.edu`, CC was 44% slower than baseline TCP, 6.7 seconds to 4.6 seconds. Analyzing the trace produced shows that the 64KB can be read by the client (which has a 20Mb/s bandwidth) in about 26 *ms*. With an RTT of about 65 *ms*, we predict the stream delay as about 26/65, which is 40%. It should be noted that as wireless network bandwidth increases (soon to potentially 108 Mb/s peak speed), the overhead from *closed acks* will drop considerably. To verify this, we ran a test to `cs.stanford.edu` with 40 Mb/s, and the slowdown was 16%, compared to a predicted slowdown of 20%.

The time overhead of PSM, unlike CC, is solely dependent on the round-trip time. Essentially, PSM rounds the RTT to the nearest multiple of 100 *ms* (see Chapter 2). Therefore, PSM is slower than baseline TCP by an average of 100%, which ranges from 511% for the lowest RTT (`cs.uiuc.edu`) to 29% for the largest RTT less than 100 *ms* (`research.microsoft.com`, at 84 *ms*); see Figure 6.5. For `inria.fr`, which has a 120 *ms* RTT, the effective RTT becomes 200 *ms*, which adds 61% overhead.

Figure 6.5 also shows that CC has lower time overhead than PSM whenever the RTT is less than 75 *ms* or greater than 100 *ms*. For a 20 Mb/s access point, the crossover point between CC and PSM is at approximately 75 *ms* if the average window size is 64KB (the maximum). This is because the stream delay due to *closed acks* will be about 25 *ms*, and PSM also increases the RTT 25 *ms*. (Under ideal conditions, CC would take slightly longer due to the delay, D, before the

window probe packet.) Note that as the average window size decreases, the crossover point will be a larger RTT.

Emulated Experiments We also experimented in an emulated environment so that we could precisely control round-trip time and loss rate. On the whole, the results are fairly similar to the Internet experiments. However, as shown in Figure 6.6, the energy consumption much more clearly trends downwards as both the RTT and loss rate increase. As described above, a large RTT means that the *closed ack* overhead is relatively small. Furthermore, the average window size decreases as loss increases, which also improves the *closed ack* overhead. Note also that here, as we are using a single server, the advertised window size is similar for a given loss rate.

Figure 6.7 compares CC to PSM and the Bounded Slowdown Protocol [59]. Comparing CC to PSM, we see that at low RTTs, CC is clearly superior to PSM in transmission time, because PSM rounds effective RTTs up to 100 *ms*. A similar effect is seen at 120 *ms*—PSM also performs poorly because the RTT is effectively 200 *ms*. As the RTT increases (but is less than 100 *ms*), PSM improves in a relative sense in both time and energy, because the effective RTT increase is smaller.

At an RTT of 90 *ms*, PSM theoretically should perform better than CC (because the effective RTT increase is small). However, PSM introduces ack compression, which sometimes eventually results in TCP decreasing its congestion window—a reaction that is due to its belief that there is a problem on the network or at the receiver. This results in the sender returning to slow start. PSM increases the time without any ack appearing at the server (after a window of packets is sent by the sender) to at least 100 *ms*. Note that BSD is identical to baseline TCP. This is because we use BSD-50, which accepts a maximum slowdown of 50%. We chose BSD-50 because CC adds less than 50% overhead in transmission time. BSD-50 operates exactly as baseline TCP at any RTT less than 200 *ms*, which was the case in all of our experiments.

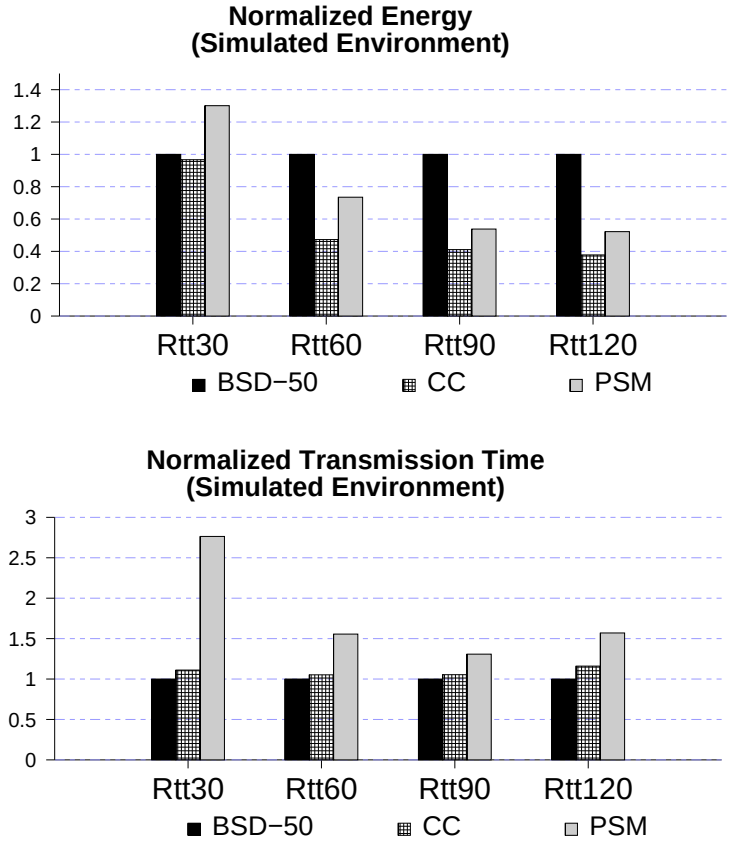

Figure 6.7: Normalized energy savings and transfer time results for CC with *active*, PSM, and BSD-50 at varying RTTs. We simulated peak hours, with a 0.1% loss rate, and a 20 Mb/s access point.

Figure 6.8 gives a breakdown of the energy consumed for an optimal algorithm, baseline TCP, PSM, and CC using the three different of end-of-burst detection algorithms. An optimal energy-conserving scheme would transition the WNIC into *receive* or *transmit* mode only to receive or transmit packets and keep the WNIC in *sleep* mode at all other times. In practice, of course, this cannot be achieved.

We observe that PSM is close to an optimal algorithm for energy consumption—all energy is in receiving packets, sending acknowledgements, and sleeping—though it slows the stream by 30%.

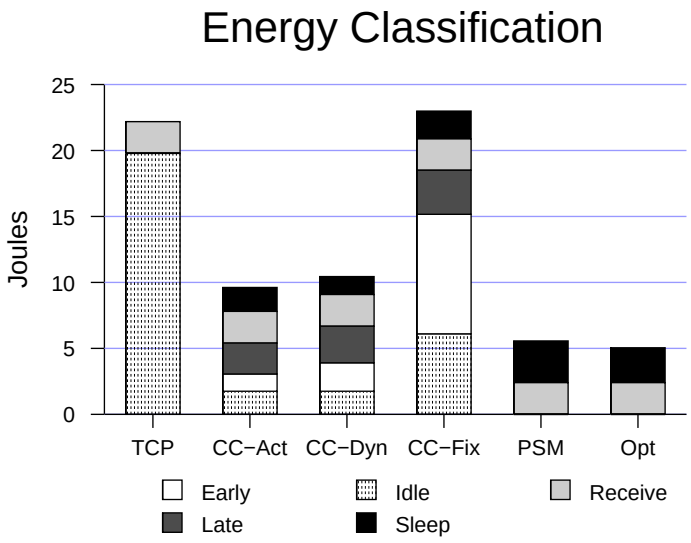

Figure 6.8: Analysis of energy consumption using CC, baseline TCP, and PSM. We simulated a 60*ms* round trip time during peak hours, with a 0.1% loss rate, and a 20 Mb/s access point.

Among the different choices for end of burst detection with CC, *active* and the dynamic threshold are competitive. Both suffer overhead due to early wakeup and waiting to transition the WNIC to *sleep* mode. The fixed algorithm, on the other hand, suffers from (much more) missed packets due to RTT variation. This causes the client to spend a significant amount of time waiting for a packet when none is in flight (counted as early wakeup).

6.3.3 Detailed Results

This section discusses two aspects of our system in detail using *DummyNet*. First, we investigate the effects of RTT variation, both on the network as well as due to access point contention. Second, we investigate the effects of access point bandwidth.

RTT Variation We first discuss the effects of variations in round-trip times. This can be variation due to the network or variation due to other clients sharing the access point. The key to handling

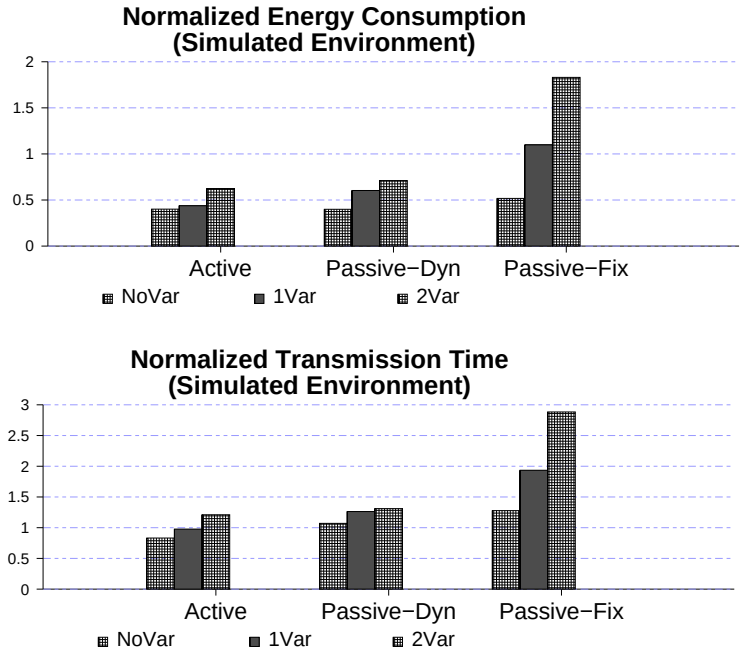

Figure 6.9: Normalized energy savings and transfer time results ranging from no RTT variation to a maximum of 50% increase over the base RTT (called *2var*) . We used a 20 Mb/s access point, a 60 *ms* RTT, and 0.1% loss.

RTT variation lies in the end-of-burst detection algorithm, as the prediction of the next window is based on the minimum RTT and so is conservative (see Section 6.2).

Figure 6.9 shows the effects of network-induced variation in RTTs. Specifically, we ran experiments with no variation, the uncorrelated gamma distribution where RTTs vary from 60 to 75 *ms*, and a distribution that was the same as the gamma distribution except the difference from the 60 *ms* base RTT is doubled (i.e., 60 to 90 *ms*). The last experiment is intended to see how our *active* performs under significant RTT variation.

While both *active* and the dynamic threshold algorithms handle RTT variation due to the network, *active* is superior in the case when there is RTT variation. This is because the window probe packet allows the client to transition the WNIC as soon as a burst has ended. When there

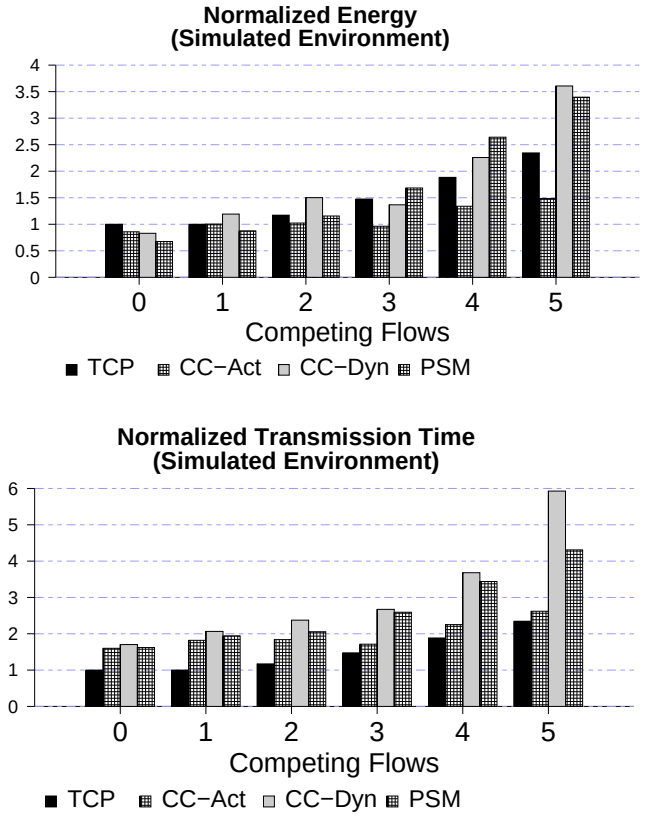

Figure 6.10: Normalized energy savings and transfer time results for a single energy-aware client along with between 0 and 5 competing regular TCP clients. The results, which are shown for *active*, dynamic threshold, and PSM, are normalized to a single TCP client. We used a 20 Mb/s access point and a 60 *ms* RTT.

is no variation, small overheads associated with *active* (e.g., energy incurred between the *open ack* and window probe packet), the dynamic threshold is slightly better.

We also experimented with multiple clients sharing an access point using CC with both *active* and a dynamic threshold. We also include a comparison to PSM. To do this, we first extended PSM to work with multiple clients. The PSM specification does not state anything about priority of packets between PSM clients and non-PSM clients. Clearly, it is not reasonable to give either type of client priority because that could lead to starvation of the other type. Therefore, we chose

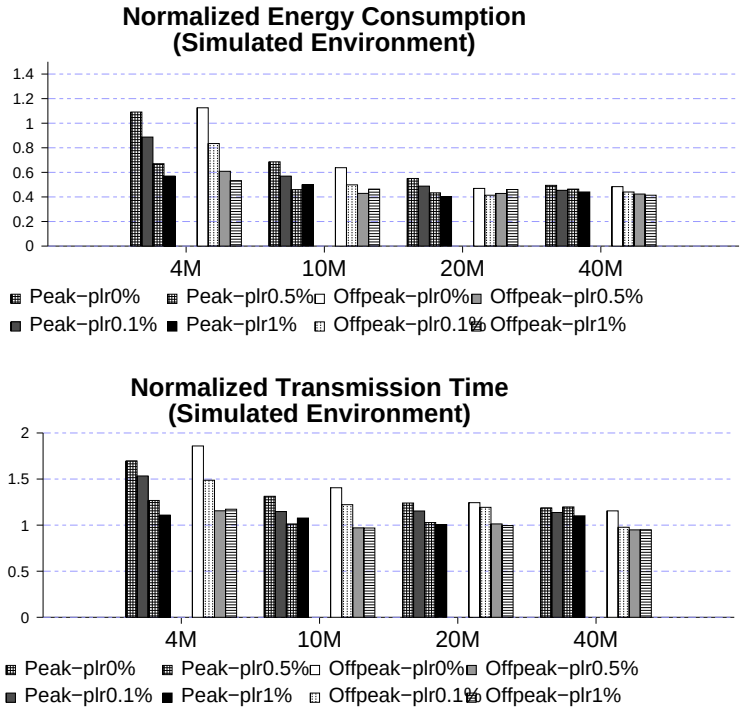

Figure 6.11: Normalized energy savings and transfer time results for comparing different bandwidth access points (4 Mb/s, 10 Mb/s, 20 Mb/s and 40Mb/s) during peak and off peak traffic. We simulated a 60ms round trip time with a 0.1% loss rate.

to use a weighted-fair-queueing algorithm at the access point when packets are being sent to both types of clients.

Figure 6.10 shows the results of a single PSM or CC client along with a variable number of competing, regular TCP clients. This experiment has all clients download the same file at the same time. Note that saturation of the access point occurs at 3 competing clients. CC with *active* consumes less energy than PSM. This is because PSM must remain in high-power mode longer due to the competing flows, whereas with CC the *sleep* time is constant (one RTT per burst).

Comparing CC with *active* to CC with a dynamic threshold indicates that *active* plays an important role in cases where multiple clients compete for access point bandwidth. The advantage

of *active* is that it takes much less time than the dynamic threshold. Again, this is because *active* can precisely determine the end of each burst—even with competing flows. Keep in mind that multiple flows can be viewed as single flows, each with RTT variation caused by the others. On the other hand, with the dynamic threshold, several packets are missed due to incorrect predictions. This also results in turn in less energy consumed with *active*. Finally, as expected, baseline TCP is faster in transmission time but consumes more energy.

Bandwidth Effects We measured the effects of different bandwidth access points: 4Mb/s, 10Mb/s, 20Mb/s, and 40 Mb/s. Figure 6.11 shows the results of our experiments with these two bandwidths during peak and off peak hours. We observe that higher bandwidth access points will result in larger energy savings and smaller time overhead. The former is because we typically have a lower link utilization and hence more time to transition the WNIC to *sleep* mode. In particular, with a 4Mb/s bandwidth, it is much more likely that the connection is saturated, which means that (1) no energy can be saved, and so (2) the client reverts to standard TCP. The latter is because the *closed ack* overhead is reduced as the bandwidth increases.

6.3.4 Effect of Bursts

We also experimentally studied the effect of bursts on routers. Specifically, we studied the impact of our energy-saving TCP flows on the bottleneck queue and standard TCP flows using an ns2 [81] simulation. Counterintuitively, we found that replacing standard TCP flows with energy-saving TCP flows *reduces* packet loss at the routers and increases the throughput of standard TCP flows.

The ns2 simulation is performed on a dumb-bell topology with a total of 100 competing TCP flows. Each flow is either a standard TCP flow or an energy-saving TCP flow. To study the impact on standard TCP flows, we measure the number of packet losses that occurred at the bottleneck, as well as the throughput of standard TCP flows. We increase the number of energy-saving TCP flows to see their impact. We also adjust the queue size of the bottleneck link to be to be 0.5, 1, and

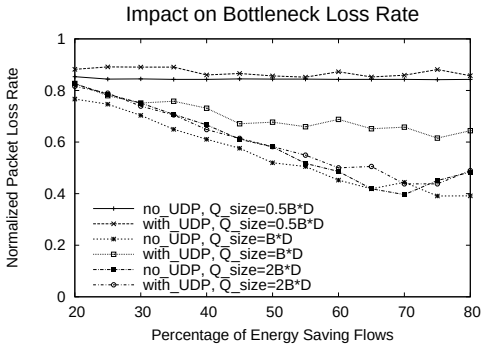

Figure 6.12: Number of packet losses seen at the bottleneck for a mix of energy-saving TCP flows and standard TCP flows. The number of losses is normalized based on the number of losses when all 100 flows are standard TCP, and when the queue (Q) size is set to the bandwidth-delay product (B*D)

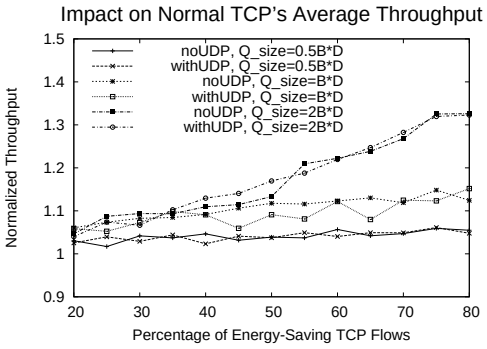

Figure 6.13: Average throughput of standard TCP flows when competing with a different number of energy-saving TCP flows. The throughput is normalized based on the average throughput of 100 standard TCP flows, and when the queue (Q) size is set to the bandwidth-delay product (B*D)

2 times the bandwidth-delay product to simulate different buffering conditions. The measurement result is counterintuitive—the loss rate of the bottleneck does not increase as the number of energy saving TCP flows increase, although these flows shape traffic into bursts. In most of the cases the loss rate actually decreases, and the average throughput of standard TCP flows increases. The loss rate result is presented in Figure 6.12, and the average throughput of standard TCP flows is presented in Figure 6.13. We also performed experiments with additional UDP traffic that introduces random transient congestion. Standard TCP flows still perform better as the number of energy-saving flows increases. By inspecting packet-level traces, we discovered that the burst size of the energy-saving TCP flows is not arbitrarily large but rather is limited by congestion control. In addition, the energy-saving TCP flows actually compete for bandwidth less aggressively than standard TCP flows (as mentioned in Section 6.2.6). As a result, as the number of energy-saving TCP flows increases, the performance of standard TCP flows improves.

6.4 Summary

In this chapter, we have explored the use of traffic shaping to create bursty transmission during large TCP downloads for energy savings. We have shown how to exploit TCP congestion control and flow control on the wireless client side to manipulate the sender to transmit data in bursts at agreed upon intervals. We also proposed and compared three end-of-window detection methods. We have conducted extensive experiments on a local emulated test environment as well as on the real Internet. With different *DummyNet* configurations, we tested our technique under a large variety of network conditions that have different RTTs, available bandwidths, and loss rates. We pointed out the potential concerns of our technique and finished with an in-depth analysis of performance.

With this chapter, we have concluded the main body of this thesis. In the next and final chapter, we summarize the work presented in this thesis, discuss our contributions, and give ideas to ways this work may be extended in the future.

Chapter 7

Conclusion and Future Work

Energy management is a critical problem in mobile computing. There is continuing evidence that it will remain a challenge as the field evolves. While there is a large research investment in low-power circuit design and hardware power management for more energy-efficient systems, there is a growing realization that more is needed– the higher levels of the system, the operating system and applications, must also contribute to energy conservation.

Although there are many power saving protocols and transmission power control techniques proposed for energy conservation on WNIC, no protocol is designed for energy efficiency in the network layer. As TCP is the most widely used transport protocol in the Internet, adapting TCP to be power aware for wireless communications is important and necessary. However, upgrading transport protocols in the global Internet is hard. This dissertation presents a detailed exploration of high-level energy management. It achieves energy efficient wireless communication techniques at the network layer. This work is an essential part of a high level comprehensive energy management strategy.

The basic idea to for our energy efficient communication technique is to reduce energy consumed by a WNIC by transitioning it to a lower-power *sleep* mode when data is not being received or transmitted. It normally requires packets to be sent in bursts at agreed-upon intervals in order to allow the WNIC to be put into a lower power-consuming *sleep* state during those intervals. The IEEE PSM mechanism achieves this goal at MAC layer with the help of the access point. In contrast, our techniques work at the higher-level network layer. It involves the tracking of each connection at the wireless client side, the shaping of traffic if necessary, and the predicting of the end of burst and the next packet arrival time. We validate our techniques in the thesis with two

dominant wireless applications, web browsing and large TCP downloads. We prove that our techniques can significantly reduce the energy usage of the WNIC without sacrifice much performance.

In the next section, we review the specific contributions of this dissertation in more detail. Then, in Section 7.2, we discuss some of the possible directions for future research generated by this work. Section 7.3 concludes by reviewing the major lessons that should be taken from this research.

7.1 Contributions

This dissertation makes contributions in three major aspects. We have made the first effort that attempts to achieve energy conservation with fine-grained switching between different WNIC power modes within the TCP protocol layer. This is a significant advance over. Previous MAC layer power saving techniques have a significant disadvantage in that they must operate completely at the link layer with no higher-layer knowledge. As a result, they cannot exploit upper layer connection or application information. Their solutions generally exhibit poor flexibility and adaptability; they are either too fine-grained for some applications or too coarse-grained for others. For example, IEEE 802.11 PSM with a 100ms beacon period is too coarse-grained to save energy during web browsing, yet too fine-grained that wastes energy for receving beacons during user think time. Although the BSD protocol can adapt during user think time, it leaves the coarse-grained problem unsolve and cannot save energy during active web browsing. More important, it is hard to deploy any upgrade in MAC layer because it involves the re-engineering of the entire network traffic.

On the other hand, energy-aware adaptations at the application level (that do not exploit operating system level information) face the problem that they are too coarse grained in directing the transition of low level WNIC power modes. Also, the lack of awareness between applications also make their attempts hard to be effective when multiple applications exist.

Our technique operates at network layer and manages all the connections in the wireless client. By actively tracking each connection, our technique can successfully exploit the characteristics of the TCP connection for energy savings. It can be adapted for different upper layer applications by detecting application port number.

The second contribution is that our technique is *client-centered*. Our technique allows mobile clients to save energy in a client-centered manner, i.e., *without* any assistance from servers, proxies or IEEE 802.11b power saving mode (PSM) in the access point. It is hard to upgrade network protocols in the global Internet because it is necessary for both communication end points to be upgraded. The lag caused by standardization, implementation by vendors, and widespreaded deployment is usually measured in years. A more complicating factor is that new protocols typically must be implemented by operating system vendors, not third parties, because network protocols reside in the kernel.

Our implementation works with client-side modifications to TCP, and so it can be easily and incrementally deployed through individual end host upgrades. Our technique solely depends on the local network at the client side, it does not need any extra assistance from the underlying network infrastructure such as intermediate routers, proxies, or access points.

The final contribution is the experimental methodology and results that validate the ideas presented in the dissertation. Our implementation uses loadable kernel modules. They become effective once loaded and does not need the rebuilding of operating system kernel at the client side. Combined with our client-centered feature, we can run the designated experiments immediately in the real Internet without requiring upgrades on the remote servers. In this manner, we run *actual experiments* to real Internet servers to validate our technique, as opposed to just simulations. Although simulation is a common way to conduct networking research, it is hard to investigate the effect of real Internet dynamics. As a result, some protocols that validated through simulations do not achieve the same effect when applied in the Internet. For this reason, some large scale real Internet testbeds such as PlanetLab [90] and Emulab [111] are provided for network researchers

to conduct experiments on the real Internet. We prove the effectiveness of our technique under real Internet traffic and obtain insights on the Internet dynamics. We believe our work is a step in pushing network experiments to include real Internet validation, if possible. This helps the network protocols improve their abilities to deal with real Internet dynamics.

7.2 Future Work

Mobile computing is rapidly evolving and has imposed new requirements to network protocols for supporting performance concerns such as energy, security, and mobility. The design, analysis, and deployment of those protocols for mobile computing has become an active research area. In the rest of this section, I discuss some of the interesting directions for future research based on this disseration.

7.2.1 Energy Efficient Systems

Energy management remains a relatively new topic in systems research. Consequently, there are opportunities for energy efficient optimizations in every component in network infrastructure. To date, most of the work in energy-efficient network research has focused on improving energy efficiency at the wireless MAC layer. This leaves the exploit of packet scheduling at the intermediate router largely untouched. Actually, it could be of great benefit to wireless clients if the infrastructure network is power aware. In other words, the future solution should be that all network components cooperate together to support the more aggressive power saving strategies.

At another dimension, with energy conservation protocols proposed at every layer in a system from low-level MAC layer to high-level network and application layer, it is desirable to exploit the increased cross-layer information awareness to support better energy conservation. One current on-going project, client-directed adaptive PSM, investigates how a wireless client can use network layer information to actively direct the operation of MAC layer PSM for power savings to conserve

energy with negligible overhead. This solution is effective for Web sites with a large number of concurrent connections or with short RTTs. It also avoids droping packets because it takes advantage of PSM buffering at the access point.

Tailoring client-centered power saving techniques for diverse upper layer applications such as online streaming, chatting, and gaming is another research area I am likely to explore. As mobile applications and services proliferate, they pose different QoS requirements when underlying power conservation techniques attempt to trade off performance for energy. Tailoring and adapting underlying power saving techniques for upper layer applications can yield the most benefit when meeting application specific requirements.

Some possible research topics listed here are:

- *Energy Efficient Wireless Communication Protocols*
- *Power-Aware Packet Scheduling*
- *Building Better AP for Power Savings*
- *Power-Aware Congestion Control in Mobile Ad hoc Networks(MANETs)*
- *The "Greening" of the Internet*

7.2.2 Network Protocols for Mobile Computing

TCP/IP is the most widely used network protocol. Current TCP implementations are optimized mainly for fair bandwidth sharing and high throughput, which is achieved by the cooperation of all TCP peers in the Internet. The concerns of the degraded effectiveness of TCP due to diverse requirements in mobile computing environments has stimulated research for better and compatible network protocols.

This dissertation has focused on the design of power-aware transport protocols. It can be naturally extended to address some other active research issues listed here.

- **Client-Centered Techniques for Diverse Applications** The client-centered technique is an attractive way for designing, testing and upgrading network protocols. I am motivated to extend the client-centered technique to design protocols to meet the diverse QoS requirements from applications such as online gaming, chatting, and streaming.

- **Network Support for Security and Mobility** I am planning to expand my research to address the practicality and the effectiveness of wireless attacks. Particularly, I will investigate how network level techniques with cross-layer information can mitigate wireless vulnerabilities. I am also interested in techniques that integrates TCP layer methods such as TCP-Handoff or TCP-Migration with IP layer information to support mobility in the future.

- **Network Protocols for Mobile Ad Hoc Networks(MANET)** The two unique characteristics of MANETs, the lack of dedicated network infrastructure and the dynamic changing environment make it challenging to design efficient network protocols for route selection, congestion control and packet scheduling in MANETs. I have put some initial effort into this area and hope I can continue this line of work in the future [121].

- **Experimental Study of Network Protocols** Experimental study is my research strength and I am willing to expand it into the analysis and validation of new network protocols.

7.3 Closing Remarks

It is likely that battery energy will continue to be a significant constraint in the design of mobile systems for the foreseeable future. Since battery capacity is limited by the size and weight consideration in mobile systems, reducing system energy usage will continue to be a primary concern. This is especially true in the Mobile Ad Hoc Network (MANET) environment, where it is hard to recharge batteries in mobile nodes.

As energy becomes an important system resource, the management of it inevitably involves the tradeoffs between energy conservation and performance. Mobile systems should provide maximum flexibility by embracing these tradeoffs. In particular, mobile systems must have a set of QoS policies for energy conservation to direct the saving of energy dynamically to support energy

performance tradeoffs. For example, when energy is deficient or performance is unimportant, e.g., background download or full speed is unnecessary, systems should optimize for energy conservation. The important point is that mobile systems should have a set of mechanisms to guarantee the QoS required by the upper layer applications or users.

In the future, energy efficient should be an important goal in the design of every component in the system. Currently, low-power circuit design and hardware power managment provide only a partial solution. Aside from MAC layer power conservation protocols, network protocols can play a more active role in energy saving. One reason is that power aware network protocols are in a sense hardware independent. This is means that there are no restraints in applying the techniques. Also, power aware network protocols work at a fine granularity compared to MAC layer or hardware layer approach. More importantly, network layer protocols are coupled with upper layer applications, which means they can exploit high-level information for better energy conservation.

In this dissertation, we developed a novel method for energy efficient wireless communications at network layer in a *client-centered* manner. Our energy efficient communication techniques provide the ability to realize the goal of network layer support for energy conservation. It is essentially a significant part of the work for the comprehensive energy conservation in mobile systems. We believe that energy efficient communication techniques will be a vital component of a complete energy management solution in future mobile systems.

Bibliography

[1] Conserving transmission power in wireless ad hoc networks. In *Proceedings of the Ninth International Conference on Network Protocols (ICNP'01)*, page 24. IEEE Computer Society, 2001.

[2] Anurag Acharya and Joel Saltz. A study of internet round-trip delay. Technical Report CS-TR-3736, 1996.

[3] Alexa Top Sites. `http://www.alexa.com/site/ds/top_500`.

[4] Virgilio Almeida, A. Bestavros, M. Crovella, and A. Oliveira. Characterizing reference locality in the WWW. In *Proceedings of PDIS*, December 1996.

[5] Manish Anand, Edmund Nightingale, and Jason Flinn. Self-tuning wireless network power management. In *Mobicom*, September 2003.

[6] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. In *Proceedings of the 5th ASPLOS*, 1992.

[7] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.

[8] Hari Balakrishnan, V.N. Padmanabhan, and R.H. Katz. The effects of asymmetry on TCP performance. In *Mobicom 1997*, Atlanta, GA, Sep 1997.

[9] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS*, May 1998.

[10] R.T. Braden. Extending tcp for transactions–concepts, Nov 1992.

[11] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP vegas: New techniques for congestion detection and avoidance. In *SIGCOMM*, pages 24–35, 1994.

[12] Eddy Caron, Olivier Cozette, Dominique Lazure, and Gil Utard. Virtual memory management in data parallel applications. In *HPCN Europe*, pages 1107–1116, 1999.

[13] S. Chandra and A. Vahdat. Application-specific network management for energy-aware streaming of popular multimedia formats. In *USENIX Annual Technical Conference*, 2002.

[14] Surendar Chandra. Wireless network interface energy consumption implications of popular streaming formats. In *Multimedia Computing and Networking (MMCN '02)*, Jan 2002.

[15] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wirel. Netw.*, 8(5):481–494, 2002.

[16] Jyh-Cheng Chen, Krishna M. Sivalingam, and Prathima Agrawal. Performance comparison of battery power consumption in wireless multiple access protocols. *Wirel. Netw.*, 5(6):445–460, 1999.

[17] Yu-Chung Cheng, Urs Hlzle, Neal Cardwell, Stefan Savage, and Geoffrey M. Voelker. Monkey see, monkey do: A tool for tcp tracing and replaying. In *Usenix*, June 2004.

[18] C. F. Chiasserini and R. R. Rao. Pulsed battery discharge in communication devices. In *Proc. 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM '99)*, 1999.

[19] Wayne A. Christopher, Steven J. Procter, and Thomas E. Anderson. The Nachos instructional operating system. Technical Report CSD-93-739, University of California, Berkeley, January 1993.

[20] IEEE Computer Society LAN/MAN Standards Committee. IEEE Std 802.11: Wireless LAN medium access control and physical layer specification. August 1999.

[21] Intel Corporation. Broadband wireless technology. Jul 2004.

[22] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations, Jan 1999.

[23] A. Datta, A. Celik, J. G. Kim, D. E. VanderMeer, and V. Kumar. Adaptive broadcast protocols to support power conservant retrieval by mobile users. In *ICDE*, pages 124–133, 1997.

[24] V. Delaluz, A. Sivasubramanian, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Scheduler-based DRAM energy management. In *Proc. Design Automation Conf. (DAC '02)*, Jun 2002.

[25] F. Douglis, P. Krishnan, and B. Bershad. Adaptive disk spin-down policies for mobile computers. In *Proc. 2nd USENIX Symp. on Mobile and Location-Independent Computing*, 1995.

[26] J. Ebert, B. Stremmel, S. Eckhardt, and W. Adam. An energy-efficient power control approach for wlans. *Journal of Communications and Networks (JCN)*, 2(3):197–206, September 2000.

[27] J. Ebert, B. Stremmel, S. Eckhardt, and W. Adam. Combined tuning of rf power and medium access control for wlans. *Mob. Netw. Appl.*, 6(5):417–426, 2001.

[28] Carla S. Ellis. The case for higher-level power management. In *Proc. 7th Workshop on Hot Topics in Operating Systems*, Mar 1999.

[29] Hari Balakrishnan et al. Improving tcp/ip performance over wireless networks. In *Mobicom 1995*, Atlanta, GA, September 1995.

[30] Hari Balakrishnan et al. Improving TCP/IP performance over wireless networks. In *Mobicom 1995*, Atlanta, GA, November 1995.

[31] Laura Marie Feeney and Martin Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *IEEE INFOCOM*, 2001.

[32] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Symposium on Operating Systems Principles*, pages 48–63, December 1999.

[33] S. Floyd. TCP and explicit congestion notification. *ACM Computer Communication Review*, 24(5):10–23, 1994.

[34] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[35] Jim Gettys and Henrik Frystyk Nielsen. The WebMUX protocol. Internet Engineering Task Force, 1998.

[36] Gnutella.

[37] K. Govil, E. Chan, and H. Wasserman. Comparing algorithm for dynamic speed-setting of a low-power CPU. In *Mobile Computing and Networking*, pages 13–25, 1995.

[38] Robyn Greenspan and Mark Berniker. Hotspot market heats up. Jul 2003.

[39] The Open Group. Business scenario: The executive on the move. Jul 2002.

[40] K.P. Gummadi, S. Saroiu, and S.D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *SIGCOMM IMW 02*, February 2002.

[41] Jiang Hao and Constantinos Dovrolis. Source-level IP packet bursts: Causes and effects. In *ACM Internet Measurement Conference 2003*, Oct 2003.

[42] Paul J. M. Havinga. *Mobile Multimedia Systems*. PhD thesis, Univ. of Twente, Feb 2000.

[43] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *Mobile Computing and Networking*, pages 130–142, 1996.

[44] J. C. Hoe. Improving the start-up behavior of a congestion control sheme for TCP. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, volume 26,4, pages 270–280, New York, 26–30 1996. ACM Press.

[45] Cisco Systems Inc. Quick reference guide cisco aironet 340 series products, January 2001.

[46] V. Jacobson, R. Braden, and D. Borman. Rfc 1323: Tcp extensions for high performance, May 1992.

[47] V. Jacobson, R. Braden, and D. Borman. Internet draft draft-ietf-tcplw-high-performance-00.txt. In *IETF*, February 1997.

[48] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.

[49] Manish Jain and Constantinos Dovrolis. End-toend available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *SIGCOMM 02*, February 2002.

[50] R. Jain. A delay based approach for congestion avoidance in interconnected heterogeneous computer networks. *Computer Communications Review, ACM SIGCOMM*, pages 56–71, 1989.

[51] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring tcp connection characteristics through passive measurements. In *Infocom 04*, February 2004.

[52] J.Bellard and S. Savage. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *Proceedings of the USENIX Security Symposium*, August 2003.

[53] Hao Jiang and Constantinos Dovrolis. Passive estimation of tcp round-trip times. In *ACM CCR*, July 2002.

[54] Eun-Sun Jung and Nitin Vaidya. A power control mac protocol for ad hoc networks. In *Mobicom 2002*, Atlanta, GA, September 2002.

[55] Kansas Unversity Real-Time Linux. `http://www.ittc.ku.edu/kurt/`.

[56] Kazza. `http://www.kazza.com`.

[57] K.Brown and S.Singh. M-TCP: TCP for mobile cellular networks. In *ACM Computer Communications Review*, Atlanta, GA, Sep 1997.

[58] David Kotz and Kobby Essien. Analysis of a campus-wide wireless network. In *Mobicom*, pages 107–118, September 2002.

[59] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Mobicom 2002*, Atlanta, GA, September 2002.

[60] R. Kravets and P. Krishnan. Power management techniques for mobile communication. In *Mobicom 98*, pages 157–168, Oct 1998.

[61] R. Kravets and P. Krishnan. Application-driven power management for mobile communication. In *ACM Wireless Networks*, Nov 2000.

[62] R. Kravets, K. Schwan, and K. Calvert. Power-aware communication for mobile computers. In *Proc. 6th International Workshop on Mobile Multimedia Communications*, Nov 1999.

[63] Long Le, Jay Aikat, Kevin Jeffay, and F. Donelson Smith. The effects of active queue management on web performance. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 265–276. ACM Press, 2003.

[64] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power aware page allocation. In *Architectural Support for Programming Languages and Operating Systems*, pages 105–116, 2000.

[65] K. Li, R. Kumpf, P. Horton, and T. E. Anderson. A quantitative analysis of disk drive power management in portable computers. In *USENIX Winter*, pages 279–291, 1994.

[66] Cirrus Logic. Whitecap2 wireless network protocol — white paper. Technical Report, 2001.

[67] Guohan Lu and Xing Li. On the correspondency between tcp acknowledgment packet and data packet. In *ACM Internet Measurement Conference 2003*, Oct 2003.

[68] S. Lu, V. Bharghavan, and R. Srikant. Fair queueing in wireless packet networks. In *ACM SIGCOMM '97*, Sep 1997.

[69] Bruce A. Mah. An empirical model of HTTP network traffic. In *INFOCOM (2)*, pages 592–600, 1997.

[70] M.Allman and V. Paxson. On estimating end-to-end network path properties. In *SIGCOMM 99*, February 99.

[71] Thomas L. Martin and Daniel P. Siewiorek. Balancing batteries, power, and performance: system issues in cpu speed-setting for mobile computing, 1999.

[72] M.C.Chan and R.Ramjee. TCP/IP performance over 3G wireless links with rate and delay variation. In *Mobicom 2002*, Atlanta, GA, Sep 2002.

[73] Leslie Jill Miller. The iso reference model of open systems interconnection: A first tutorial. In *Proceedings of the ACM '81 conference*, pages 283–288. ACM Press, 1981.

[74] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *Proceedings of the 16th international conference on Supercomputing*, pages 35–44, 2002.

[75] Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean C. Walrand. Analysis and comparison of TCP reno and vegas. In *INFOCOM (3)*, pages 1556–1563, 1999.

[76] J. Monks. Transmission power control for enhancing the performance of wireless packet data networks, 2001.

[77] Jeffrey P. Monks, Jean-Pierre Ebert, Adam Wolisz, and Wen mei W. Hwu. A study of the energy saving and capacity improvement potential of power control in multi-hop wireless networks. In *Proceedings of the 26th Annual IEEE Conference on Local Computer Networks*, page 550. IEEE Computer Society, 2001.

[78] A. Mukherjee. On the dynamics and signicance of low frequency components of internet load. *Internetworking: Research and Experience*, 5(4):163–205, December 1994.

[79] Thyagarajan Nandagopal, Tae-Eun Kim, Xia Gao, and Vaduvur Bharghavan. Achieving mac layer fairness in wireless packet networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 87–98. ACM Press, 2000.

[80] Netfilter. `http://www.netfilter.org`.

[81] Network Simulator-2. `www.isi.edu/nsnam/ns/`.

[82] NSWEB. `www.net.uni-sb.de/~jw/nsweb/`.

[83] 802.11 Standards Committee of the IEEE Computer Society. Ieee standard 802.11, 1999 edition, 1999.

[84] LAN/MAN Standards Committee of the IEEE Computer Society. Part 11: Wireless medium access control (mac) and physical layer (phy) specifications, 1999.

[85] LAN/MAN Standards Committee of the IEEE Computer Society. Part 11: Wireless medium access control (mac) and physical layer (phy) specifications: Higher-speed physical layer extension in the 2.4ghz band, 1999.

[86] Jitendra Padhye and Sally Floyd. On inferring tcp behavior. In *SIGCOMM 01*, February 2001.

[87] Athanasios E. Papathanasiou and Michael L. Scott. Increasing disk burstiness for energy efficiency. Technical Report 792, University of Rochester, November 2002.

[88] Athanasios E. Papathanasiou and Michael L. Scott. Energy efficiency through burstiness. In *WMCSA*, October 2003.

[89] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *ISLPED 1998*, August 1998.

[90] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of HotNets–I*, Princeton, New Jersey, October 2002.

[91] Larry Peterson and Bruce S. Davie. Computer networks: A systems approach. In *Morgan Kaufmann*, 2003.

[92] J. Postel. Transmission control protocol, 1981.

[93] Johan Pouwelse, Koen Langendoen, and Henk Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 251–259. ACM Press, 2001.

[94] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Infocom 2002*, February 2002.

[95] Luigi Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM Computer Communications Review*, 27(1), January 1997.

[96] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[97] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. An analysis of internet content delivery systems, 2002.

[98] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. TCP congestion control with a misbehaving receiver. *Computer Communication Review*, 29(5), 1999.

[99] Prashant Shenoy and Peter Radkov. Proxy-assisted power-friendly streaming to mobile devices. In *MMCN*, Santa Clara, CA, January 2003.

[100] Eugene Shih, Paramvir Bahl, and Michael Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Mobicom 2002*, September 2002.

[101] Tajana Simunic, Luca Benini, Peter Glynn, and Giovanni De Micheli. Dynamic power management for portable systems. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 11–19. ACM Press, 2000.

[102] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Mobile Computing and Networking*, pages 181–190, 1998.

[103] M. Stemm, P. Gauthier, D. Harada, and R. H. Katz. Reducing power consumption of network interfaces in hand-held devices. In *Proc. 3rd Intl. Workshop on Mobile Multimedia Comm.*, September 1996.

[104] Intel Toshiba, Compaq. The advanced configuration and power interface specification. Jul 1998.

[105] J. Touch. Tcp control block interdependence, Apr 1997.

[106] A. Vahdat, A. Lebeck, and C. Ellis. Every joule is precious: The case for revisiting operating system design for energy efficiency. In *Proc. 9th ACM SIGOPS European Workshop*, 2000.

[107] Zhiheng Wang, Amgad Zeitoun, and Sugih Jamin. Challenges and lessons learned in measuring path for proximity-based applications. In *PAM 2003*, February 2003.

[108] Yong Wei, Surendar Chandra, and Suchendra M. Bhandarkar. A statistical prediction-based scheme for energy aware multimedia data streaming. In *WNCN 2004*, August 2004.

[109] M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Operating Systems Design and Implementation (OSDI '94)*, pages 13–23, 1994.

[110] Richard Wendland. "How prevalent is timestamp options and paws". Web survey result, end-to-end interest list, 2003.

[111] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[112] John Wilkes. Predictive power consumption. Technical Report HPL-CSP-92-5, Hewlett-Packard Labs, Feb 1992.

[113] Gary Wright and Richard Stevens. Tcp/ip illustreated. Addison-Wesley Professional Computing Series, 1995.

[114] Hai Tao Wu, Yu Lin, Shi Duan Cheng, Yong Peng, and Ke Ping Long. Ieee 802.11 distributed coordination function: enhancement and analysis. *J. Comput. Sci. Technol.*, 18(5):607–614, 2003.

[115] Kaixin Xu, Mario Gerla, Lantao Qi, and Yantai Shu. Enhancing tcp fairness in ad hoc wireless networks using neighborhood red. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 16–28. ACM Press, 2003.

[116] Haijin Yan, Rupa Krishnan, Kang Li, Scott A. Watterson, and David K. Lowenthal. A theoretical and experimental study of energy-saving mechanisms for TCP downloads. Technical report, University of Georgia, February 2004.

[117] Haijin Yan, Rupa Krishnan, Scott A. Watterson, and David K. Lowenthal. Client-centered energy savings for concurrent HTTP connections. In *NOSSDAV*, June 2004.

[118] Haijin Yan, Rupa Krishnan, Scott A. Watterson, David K. Lowenthal, and Kang Li. Client-centered energy savings for TCP downloads. Technical report, University of Georgia, February 2004.

[119] Haijin Yan, Rupa Krishnan, Scott A. Watterson, David K. Lowenthal, Kang Li, and Larry L. Peterson. Client-centered energy and delay analysis for TCP downloads. In *Proceedings of the 14th IEEE Int'l Wkshop on Quality of Service*, June 2004.

[120] Haijin Yan, Rupa Krishnan, Scott A. Watterson, David K. Lowenthal, Kang Li, and Larry L. Peterson. Client-centered energy savings for TCP downloads. Tr, University of Georgia, February 2004.

[121] Haijin Yan and David K. Lowenthal. Towards cooperation fairness in mobile ad hoc networks. In *Proceedings of the WCNC 2005*, March 2005.

[122] Luqing Yang, Winston Seah, and Qinghe Yin. Improving fairness among tcp flows crossing wireless ad hoc and wired networks. In *MobiHoc '03*, Jun 2003.

[123] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *ASPLOS X*, October 2002.

[124] M. Zhang, B. Karp, S. Floyd, and L. Peterson. Rr-tcp: A reordering-robust tcp with dsack. In *ICNP 2003*, pages 255–270, Atlanta, GA, December 2003. USENIX Association.

[125] Y. Zhang, L. Breslau, V.Paxson, and S.Shenker. On the characteristics and origins of internet flow rates. In *SIGCOMM 2002*, February 2002.