Adaptive Channels for Wireless Networks

by

Andrew G. Chiu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering

and

Bachelor of Science in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

May 1999

Author . . . .
Department of Electrical Engineering and Computer Science
May 21, 1999

Certified by . . . .
John V. Guttag
Professor, Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Adaptive Channels for Wireless Networks
by
Andrew G. Chiu

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 1999, in partial fulfillment of the
requirements for the degrees of
Master of Engineering

and

Bachelor of Science in Electrical Engineering and Computer Science

Abstract

This thesis presents the design, implementation, and analysis of an adaptive wireless network that is capable of dynamically modifying the physical layer of its wireless links. Nodes in such a network are able to change many aspects of the physical layer, including coding, modulation, and multiple access. With this technology, a wireless network can dynamically adapt its physical layer to changing environmental conditions, traffic patterns, and application requirements, resulting in better spectrum utilization, power consumption, and application level performance.

The approach presented in this thesis overcomes many of the limitations imposed by today's wireless networks. Software radio technology is exploited to provide the necessary flexibility in the physical layer. Since changes to the physical layer must be performed quickly and reliably in a lossy wireless environment, a modification mechanism, including a reliable protocol, was designed and implemented. Putting these two technologies together, the end result is the implementation of an adaptive wireless network infrastructure that is capable of modifying its physical layer to increase performance. While the issue of the decision rules that are used to determine what changes are necessary is beyond the scope of this thesis, the presented infrastructure is general enough to support any method of determination.

Thesis Supervisor: John V. Guttag
Title: Professor, Computer Science and Engineering

Acknowledgments

I would like to show my gratitude to the following people for their invaluable contributions, without whom I could not have reached this point in my life:

- Vanu Bose, for taking a chance on an unproven freshman and being an awesome supervisor throughout my time at the lab. I am extremely grateful for his guidance, assistance, and insight into SpectrumWare, technology, sports, and life in general.
- John Guttag, for finding the time in his busy schedule to provide his guidance, advice, and support.
- Cindy Liang, for providing the love, encouragement, and understanding that helped me through.
- Michelle Girvan, for your friendship, advice, and all of the intellectual and mathematically-inclined discussions during our years at MIT.
- And finally, my family, for all of your love and support over the years and for providing me with the opportunity to pursue my dreams.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of wireless networking is to provide the same high performance of a wired network while allowing the added benefit of mobility. Today's wireless networks do not achieve this goal. Users of wireless networks experience intermittent connectivity, low bandwidth, occasional dropped connections, restricted mobility, and poorer overall performance.

One cause of these performance limitations in existing wireless networks is the static functionality at the physical layer. A wireless network today is constructed with one, predetermined physical layer. This inflexibility leads to inefficient use of resources, such as bandwidth and power, and sub-optimal performance.

1.1 Why Adaptive Physical Layers

By using adaptive physical layers, flexibility can be introduced into the physical layer to improve performance. The physical layer can then be modified for changing needs and requirements. Thus, adaptive physical layers provide the functionality to allow better use of the spectrum, allow mobility between different wireless networks, and increase overall performance.

Since the wireless links in today's networks are designed for the worst case and static, a network is unable to take advantage of better-than-worst case conditions. Link performance is thus upper bounded by the characteristics of the physical layer and not by current environmental conditions. For example, suppose that the noise level in the environment decreases, increasing the signal-to-noise ratio. This event increases the potential maximum throughput of a given channel. However, the network is unable to change its channel coding or filtering to increase throughput since it cannot modify its physical layer. Static networks are also susceptible to interference localized to certain critical frequencies. For example, the operation of a wireless network using the 2.4 GHz ISM band can be disrupted by a microwave oven, which operates at about 2.45 GHz [6]. Such a network could potentially increase performance by modifying its physical layer to avoid this localized interference by using the portions of the ISM band above and below the interference instead of the entire band.

There is no direct connection between communication link performance and application requirements in today's wireless networks. The design of the physical link provides a set of operational parameters, such as latency and bit error rate, that cannot be changed. Different applications, however, have different requirements. For example, an application transferring data can tolerate some latency, but it wants a bit error rate as close to zero as possible. On the other hand, a real-time video application, such as videoconferencing, can tolerate occasional bit errors, which results in occasional pixel errors or dropped video frames, but requires low latency. These applications want Quality of Service (QoS) at the physical layer that cannot be provided by today's systems. One solution is to subdivide the available band for each type of data. A slice of the spectrum is dedicated to video, where bandwidth is guaranteed to the application, and the rest of the band is shared among all of the users transferring data. Each subdivision has a different physical layer that meets the service requirements of the application it serves.

There are many possible applications of wireless networking technology that have not been realized because of the inconsistent performance of current networks. Because of the limitations imposed by the static nature of current implementations, the proliferation of wireless networking and its promise of anytime connectivity has been hindered. For example, a possible architecture for wireless networking is for a mobile node to be connected to several different overlay networks, each of which applies to a different coverage area [15, 16]. The smallest coverage area is a building-area network, and the others, in order of increasing size, are a campus-area, metropolitan-area, and a regional-area network. Each network is characterized by different parameters such as bandwidth, latency, and bit-error rates [15, 16]. Without the ability for a wireless node to dynamically switch between multiple physical layers, such an architecture must be constructed with the same physical layer at all levels of the overlay network system, or a node is forced to carry several different wireless network adapters. Even though this restriction does not prevent the use of overlay networks, it greatly hinders its introduction into commercial use.

1.2 Creating Adaptive Physical Layers

The key enhancement of an adaptive wireless network is a flexible physical layer. This allows the network to optimize at the physical layer in addition to the network and link layers. Information may be passed between the different communication layers (physical, link, network, and application), allowing cross-layer optimization. For example, information about bit error rates, determined at the physical layer, may be used to adjust packet sizes, which is a network layer parameter. Conversely, packet loss tolerances for applications may be used to adjust bandwidth usage and channel coding, which are both physical layer parameters. With the flow of information between the layers, optimizations improve overall system performance rather than individual layer performance.

There are three issues that must be addressed to effectively utilize adaptive physical layers. The first is the decision on when a modification is necessary and what change is to be implemented. Functionality needs to be built into the network to detect changes in the environment or user requirements and to determine which modifications to the wireless channels are necessary to improve performance. This issue, while not the focus of this thesis, is addressed as future work in chapter 6. The second issue is the execution of these modifications, and the third issue is the coordination between the two ends of the wireless link to ensure quick and reliable modifications to the physical layer. These two issues are covered in this thesis, and software radio technology provides the flexibility that is required to address them.

1.2.1 The Flexibility of Software Radio Technology

Software radio technology has the promise to provide two levels of flexibility: static and dynamic. Static flexibility is the ability to modify the physical layer at setup time. However, once the radio is running, control over the physical layer is limited. Dynamic flexibility is the ability to modify the physical layer at runtime while communication is occurring. In this case, modifications to the physical layer occur automatically and transparently to the user with minimal interruption in service.

The ability for software radios to perform static modifications has already been demonstrated. The Speakeasy Multiband Multimode Radio uses programmable processing to emulate more than 15 existing military radios, operating in frequency bands between 2 and 200 MHz [13]. In addition, many applications in the SpectrumWare project at MIT deal with setup-time modifications. The same device, using the same hardware, can be set up to be an analog TV receiver, an AMPS cellular receiver, or a wireless network interface just by changing the code [3].

Software radio technology also provides a mechanism for overcoming the limitations of current wireless networks by permitting dynamic modification of the physical layer [3]. Software control over the physical layer has been demonstrated, and the extension to dynamic "on-the-fly" adaptation is the next step. To support this rapid, run-time adaptation, a mechanism is needed to ensure quick, coordinated modifications at both ends of the wireless link.

1.2.2 Coordination of Physical Layer Modifications

The software radio-based adaptive wireless node uses the current physical layer to transmit the changes required to construct the new physical layer. Thus, it is important that the nodes on both ends of the wireless link agree to change to a new link and set up that new link before the current link is destroyed.

Since the underlying communication channel is a wireless connection, it is inherently lossy and unreliable. Bit errors are common, and message losses are expected. Thus, a protocol is needed to handle message losses and ensure that the modification occurs rapidly even in a poor environment.

The adaptive channels described in this thesis are designed to be extremely general. The design of the particular wireless network that utilizes these adaptive channels is not fixed, and the method by which individual wireless links are modified is also not fixed. It is possible that a particular network configuration allows multiple modification requests to be issued within the network at any one time. In such a case, there must be a mechanism to handle the situation where a wireless node issues a modification request and simultaneously receives a conflicting request.

1.3 Impact of Adaptive Physical Layers on Wireless Networks

A wireless network using software radios can exert control over the physical layers of its communication links. By executing a reliable modification protocol when modifying its wireless links, a network can quickly change the communication channel with minimal interruption in service. In many cases, the latency incurred by the modification will be small enough to be unnoticeable by the user. An adaptive network that utilizes both software radios and a reliable modification protocol can achieve the following goals:

- The physical layer can be adapted to meet changing environmental conditions, improving link performance.
- The physical layer can be adapted to meet the different requirements of each application, improving the overall application performance.
- Optimizations can be made across OSI layers, resulting in better overall system performance.
- System resources, such as power and CPU utilization, can be better managed, leading to more efficient operation.

Adaptive wireless networks provide the solution to many of the problems that limit existing systems. The ability of a wireless network to adjust its operational parameters to maintain or improve the performance of the system under changing circumstances brings the idea of reliable, anytime connectivity closer to reality.

1.4 Thesis Scope

In order to build an adaptive wireless network, many new technologies need to be developed. By designing the underlying infrastructure, this thesis is the first step. To demonstrate this infrastructure, this thesis develops a protocol for the coordination of physical layer modifications and presents an implementation of a two node adaptive network.

In the implemented network, each node is equipped with a software radio-based wireless network interface device and programmed with the coordination protocol. These nodes are able to reliably perform modifications to the physical layer of the network, such as a change in the modulation. In order to assess the tradeoffs of this approach, the performance of this network is measured in terms of both the coordination protocol and the software-based network interface. By building a basic adaptive network and characterizing its performance, this thesis provides insight into the strengths and weaknesses of this software approach to adaptive wireless networking.

1.5 Contributions

The major contributions of this thesis are:

- The demonstration that software radio technology can provide dynamic flexibility in the physical layer of wireless network links.
- The design, analysis, and implementation of a reliable protocol for dynamically modifying network links.
- The implementation of an infrastructure upon which future research into adaptive wireless networks can be based.

1.6 Road Map

The next two chapters deal with the major building blocks for an adaptive wireless network. Chapter 2 describes software radios and the SpectrumWare architecture, which serves as the platform upon which our network is built. This chapter also introduces the software radio layering model, which allows any physical layer to be concisely specified. Chapter 3 introduces and analyzes a protocol that provides a reliable method for modifying the physical layer quickly and transparently to the user.

After describing the individual components of the system, chapter 4 describes the design of the network infrastructure and its implementation in the SpectrumWare architecture. Chapter 5 evaluates both the performance of individual components of the system and the overall performance of the network. Finally, chapter 6 concludes with observations and suggestions for future work.

Chapter 2

SpectrumWare Virtual Radio Architecture

A *software radio* is a radio device whose modulation waveforms are defined and generated in software, which introduces flexibility by allowing software programmability [11]. Most software radio architectures utilize application-specific digital hardware or digital signal processors under software control [5, 13]. Taking the implementation one step further, a *virtual radio* is a communications device that performs all of its digital signal processing on a commercial, off-the-shelf workstation or personal computer [3].

The SpectrumWare virtual radio system described in this chapter is ideal for developing an adaptive wireless network. Many of the properties of the SpectrumWare system are exploited in the design of the adaptive network presented in this thesis. The following section gives an overview of the SpectrumWare system and highlights the aspects of the system that are central to the design of the adaptive network. Section 2.2 describes the software radio layering model which provides a specification for any physical layer, and section 2.3 describes the importance of these technologies on adaptive wireless networks.

2.1 System Architecture

The SpectrumWare virtual radio project at the MIT Laboratory for Computer Science demonstrates the feasibility of using general purpose processors coupled with wideband digitization to implement a software radio [3]. By using a general purpose processor and a standard operating system (Linux), all of the signal processing routines can be implemented in a high-level programming language, such as C++. This type of architecture provides an excellent design and development platform to explore the many advantages of software-defined radios, including [2]:

- greater flexibility in the range of functionality that can be implemented,

- ease of portability of software between processors allowing the software radio platform to track the performance of Moore's Law, and
- tighter coupling between the application and the radio allowing for better system optimization.

The SpectrumWare architecture is an excellent platform upon which to develop and implement an adaptive wireless network. Since all of the physical layer functions are performed by software in user space, modifications to the physical layer are easily executed by changing user-level code. Also, using a general purpose platform running Linux allows integration of the wireless network with the user application and the operating system. The software-based physical layer can directly interface with the bottom of the network stack through a device driver, and it can receive and send network traffic through sockets. The first ability is central to the operation of the wireless network interface, and the second ability allows this network interface to dynamically modify its physical layer.

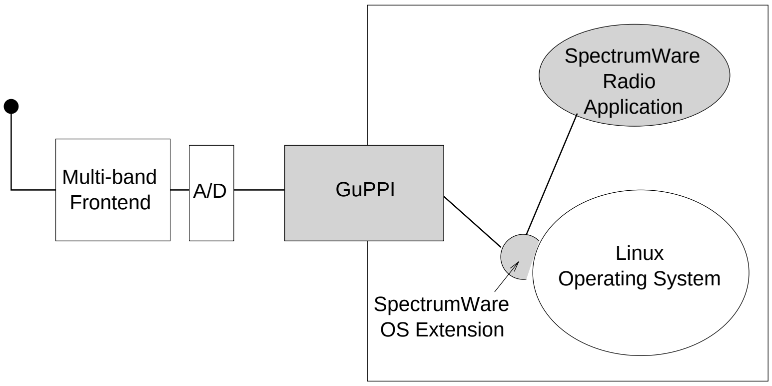

Figure 2-1: The SpectrumWare architecture.

The SpectrumWare virtual radio architecture, shown in figure 2-1, consists of two main components: the I/O system and the application programming environment. The following two sections describe each of these components.

2.1.1 I/O System

The I/O system is responsible for acquiring the frequency band of interest, digitizing it, and transporting the desired samples into host memory. The SpectrumWare system uses a multi-band frontend to convert the desired RF band to an IF frequency, samples the wideband IF waveform, and transports the resulting samples into host memory. Similarly, to transmit, the system transfers samples from memory to a D/A converter and then translates this IF waveform to the desired RF band.

The movement of samples between the A/D (and D/A) and host memory is performed by the General Purpose PCI I/O, or GuPPI [3]. It utilizes the PCI bus to perform continuous DMA between the GuPPI and host memory. However, these samples are placed in memory that is only accessible by the GuPPI device driver. To allow the radio application access to these samples, extensions were made to the operating system. Specifically, the virtual memory system was extended to provide a low-overhead, high-bandwidth transfer of data between the application and the device driver. This extension allows for copy-free read and write calls to the GuPPI, which results in a high sample throughput between the application and the GuPPI. Using a 200 Mhz Pentium Pro running Linux with a 33 Mhz, 32 bit wide PCI bus, the maximum continuous data transfer rate is 512 Mbits/sec [3].

2.1.2 The SPECtRA Programming Environment

SPECtRA, or Signal Processing Environment for Continuous Real-time Applications, provides a platform for the implementation of real-time software radio applications [2]. The system, shown in figure 2-2, consists of two partitions: the in-band data processing module chain, and the out-of-band control section.

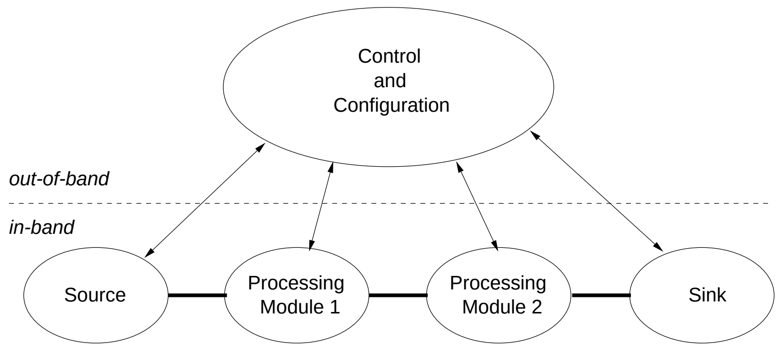

Figure 2-2: The SPECtRA environment.

In SPECtRA, all of the signal processing occurs in modules. Each module, implemented as a C++ class, performs a specific task, such as FM modulation or channel filtering. By connecting a series of modules, specific applications can be built. For example, to create a single channel FM receiver application, a possible processing chain is:

- GuPPI Source - provides a wideband sample stream from the GuPPI
- Channel Filter - selects the proper channel from the wideband IF signal
- FM Demodulator - extracts the signal/voice from the FM carrier
- Filter - isolates the frequencies present in the desired audio signal
- Audio Sink - provides an interface to the Linux audio driver

Modules are not associated with any particular application. By structuring the programming environment to be modular, code reuse is possible. Also, this modularity allows for incremental additions or changes to the processing chain to be executed easily. To modify the above FM receiver into a single channel AM receiver, the only change that is required is to replace the FM Demodulator with an AM Demodulator. The remaining modules stay the same and no other code changes are necessary.

The out-of-band control portion of SPECtRA is responsible for everything outside the actual data processing. This includes creating and modifying the in-band processing chain, executing communication between modules, and handling user interaction [2].

2.2 Software Radio Layering Model

In order to allow modifications to a wireless communication system, there must be a mechanism to specify the signal processing requirements of the new system to the radio at each end of the connection. To address this issue, the SpectrumWare virtual radio architecture uses a model consisting of several well-defined processing layers that can be used to completely specify a wireless communications system [2]. This layering, shown in figure 2-3, is a refinement of the OSI layering model [17].

Each sublayer performs a distinct function, with the bottom four software sublayers comprising the functions that are traditionally performed at the physical layer [2]:

- **Link Framing:** The traditional link layer is preserved. This is used to transform the raw transmission facility into a line that appears free of errors to the network layer [17].
- **Media Access Control (MAC):** The primary MAC functions are mediating shared medium access and collision avoidance.
- **Channel Coding:** Channel codes are used to reduce errors at the bit level through error detection, error correction, or error prevention [14].
- **Line Coding:** Line codes control the statistics of the data symbols, such as the removal of baseline drift or undesirable correlations in the symbol stream. The desired parameters are determined by physical characteristics of the transmission medium.
- **Modulation:** This sublayer deals with the transformation between symbols and signals.
- **Multiple Access:** The multiple-access sublayer implements techniques such as TDMA and FDMA. Although the MAC layer may also involve a multiple access technique, this provides a very different function. Consider the IEEE 802.11 wireless networking standard [10]. The MAC layer provides multiple access among users of a particular network, while the multiple access layer allows for the sharing of the spectrum between different networks.

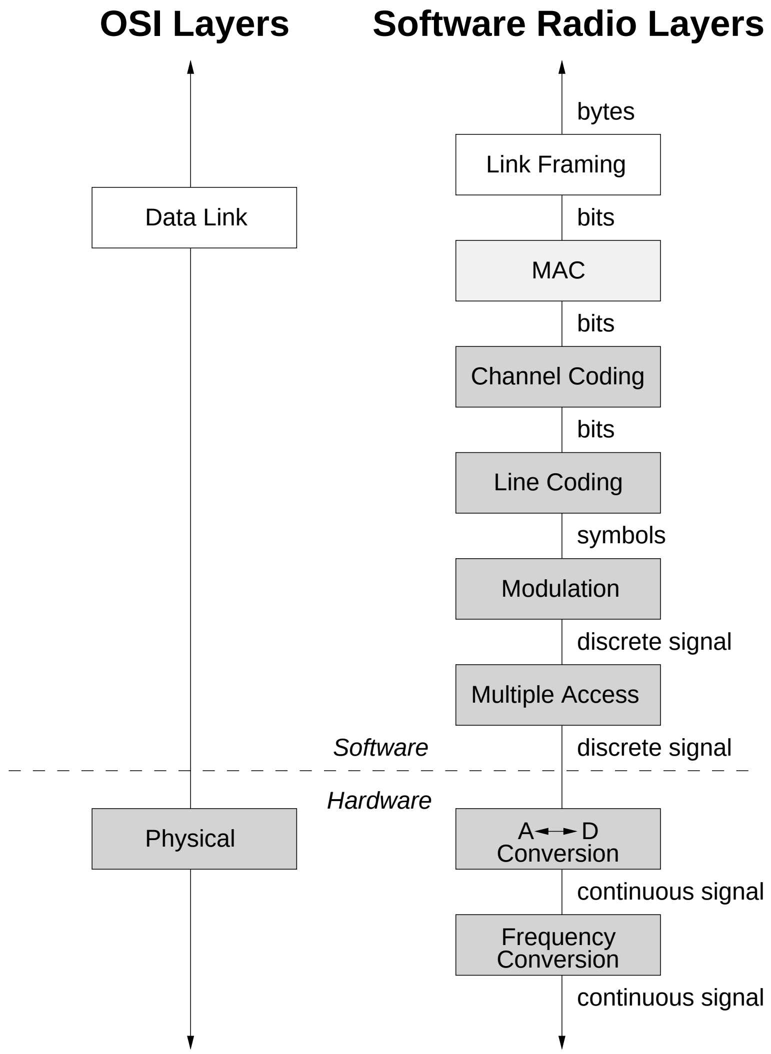

Figure 2-3: The Software Radio Layering Model shifts many physical layer functions into software. The shaded layers comprise the traditional physical layer.

This layering model provides a framework for specifying and building software radio applications. The layering provides a modular architecture in which a new communication system can be created by simply combining existing functional modules instead of writing a new piece of software that encompasses all of the required physical layer functions. Incremental changes can be specified by swapping in the desired modules and removing the modules that have been replaced. This is important, since it reduces the overhead associated with loading the new physical layer code and implementing the desired changes.

A given system may only contain a subset of the layers. However, such a system can be represented with all of the layers present, except that some of them do not manipulate the data in any way.

2.3 Application to Adaptive Networks

The SpectrumWare architecture provides an ideal platform for an adaptive wireless network. SpectrumWare, along with the software radio layering model, provides:

- **Efficient, high-bandwidth I/O.** The I/O system provides a 512 Mbits/sec transfer rate, which allows the system to process a frequency band as large as 32 MHz[1]. This bandwidth is more than sufficient for networking purposes. For example, the IEEE 802.11 standard allocates 5 MHz channels in North America and Europe and 26 MHz channels in Japan, which are both within the limitations of this I/O system [10].

- **Modular program architecture.** Functions such as filters, modulation formats, and coding techniques are written once, creating a library of reusable processing modules. Applications are then constructed from a common set of building blocks.

- **Physical layer specification and representation.** Any instance of a layer of the software radio layering model maps to one or more processing modules. Thus, any particular physical layer specified by the software radio layering model translates into a chain of processing modules, each of which is taken from the common library of reusable modules.

- **Method for adaptation.** The out-of-band control code provides a mechanism to modify the current radio application. Modifications are executed by taking the new physical layer representation and replacing the necessary processing modules. The control code can be extended to execute modifications in response to desired inputs, such as network messages or increasing error rates.

These properties allow the development of a software-based adaptive wireless network. SpectrumWare is well-suited for this application, and it provides a solid processing environment upon which the modification protocol and other intelligent code for determining physical layer changes can run.

[1] 64 Msamples/sec, 8 bits/sample

Chapter 3

Protocol for Physical Layer Modification

For a wireless network to execute a modification to the physical layer of a particular link, the specification for the new physical layer must be known to both ends of the link. Messages must be exchanged so that both ends create and quickly switch over to the same new physical layer. Also, since the wireless environment is noisy and unpredictable, an adaptive wireless network requires a reliable protocol for exchanging the necessary messages. This chapter describes one such protocol and evaluates its reliability and latency characteristics.

3.1 Communication Model

A wireless link between two hosts A and B, as shown by the example in figure 3-1, is described by two sets of parameters, one for downstream data traveling from A to B (P_d), and one for upstream data from B to A (P_u) . Each set of parameters contains one entry for each of five layers (MAC, channel coding, line coding, modulation, and multiple access) of the software radio layering model described in section 2.2. These parameters completely specify the physical layer. The two sets of parameters may be different or the same. For the case that they are the same, that one physical link may be viewed as two logical links, one for each direction. Each host transmits on its downstream link and receives on its upstream link. Thus, each host has direct control over the data transmitted in its downstream link, but not over the data in its upstream link.

The problem is as follows: while the two hosts are communicating data, one host decides to initiate a change in the physical layer, which corresponds to a change to a different set of parameters. Data transfer temporarily stops, and the host initiates the modification protocol. A message with the specifications for the new physical layer is sent to the other host. Thus, the current physical layer is used to communicate the

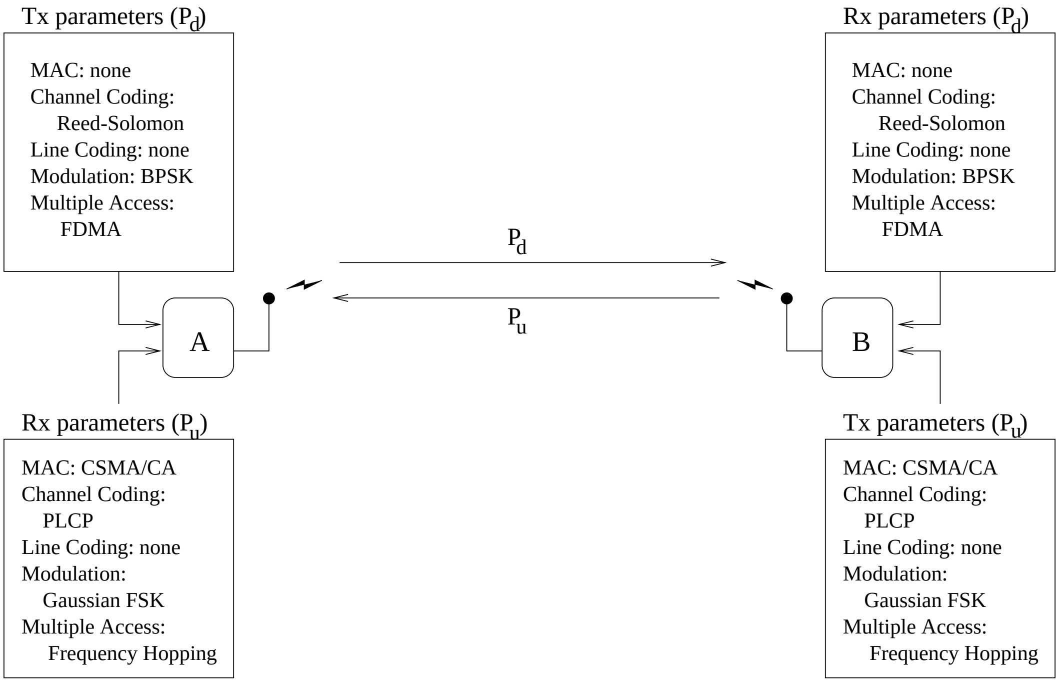

Figure 3-1: Model of the communication setup.

information necessary to create the new physical layer.

Since the underlying communication link is lossy, the protocol for executing changes to the physical layer must be reliable and able to handle message losses. Although success can never be guaranteed when losses are possible, the modification protocol must rapidly succeed with high probability in an environment with typical loss rates.

The following sections describe the mechanism by which this modification process occurs.

3.2 Basic Protocol

Let us assume that a bidirectional link has been established between the two hosts. If no link exists, a link initialization mechanism is used to establish a connection. One such mechanism is a dedicated control channel, which is used by a cellular phone system. This link initialization mechanism is also used as a fallback in the event that channel conditions degrade and communication is no longer possible over the established link. The particular choice of a link initialization mechanism is not important as long as the hosts on each end of the wireless link are using the same one.

Since the upstream and downstream links are specified by different sets of parameters (which may or may not be the same), this protocol will reconfigure one link at a time. There are two cases to consider: the host initiating the protocol may wish to modify either its downstream link or its upstream link.

3.2.1 Downstream Modification

Consider the downstream case first. Since a host has control over the data flowing over its downstream link and can stop the transmission of data over it before initiating the protocol, this case is simpler.

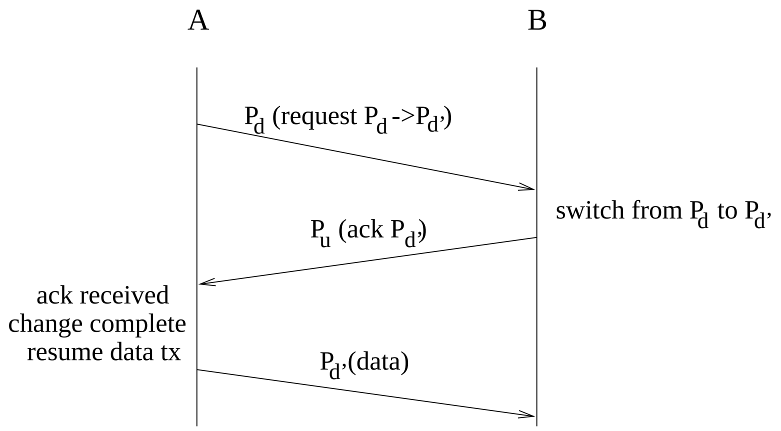

Figure 3-2: Basic protocol to modify a downstream link. Message format: *link(message)*

This protocol, shown in figure 3-2, requires the successful transmission of two messages. A typical sequence, with host A initiating a change in its downstream link, is:

- **Request.** A sends a message over P_d requesting a change from P_d to $P_{d'}$.
- **Acknowledge.** B receives the message, changes its reception code from P_d to $P_{d'}$, and sends an acknowledgment over P_u. This acknowledgment informs host A that it is ready to receive data over the new link.
- **Link established.** A receives the acknowledgement, and the new link is established. A can now resume data transmission using $P_{d'}$.

This protocol is similar to the three-way handshake used in establishing a TCP connection [8]. The difference is the underlying communication link in this protocol changes while the link in the TCP three-way handshake remains static. This change in the communication link midway through the protocol affects the handling of message losses.

If either the request or its acknowledgement is lost, host A will not receive a message back from host B. Thus, after a timeout, host A will have to retransmit the request. However, if the request is lost, host B is still listening on P_d, while if the ack is lost, B is listening on $P_{d'}$. Thus, if the timeout expires, host A will not know over which set of parameters to retransmit. The solution to this, shown in figure 3-3, is to retransmit the request on *both* channels, once on P_d and once on $P_{d'}$. Because host B is guaranteed to be listening on one of them, one of the two messages will be received if no further losses occur. If further losses do occur, the protocol continues in the same manner, with host A reaching its timeout and retransmitting on both channels. Upon receiving the acknowledgement, no further timeouts occur and data communication continues using $P_{d'}$.

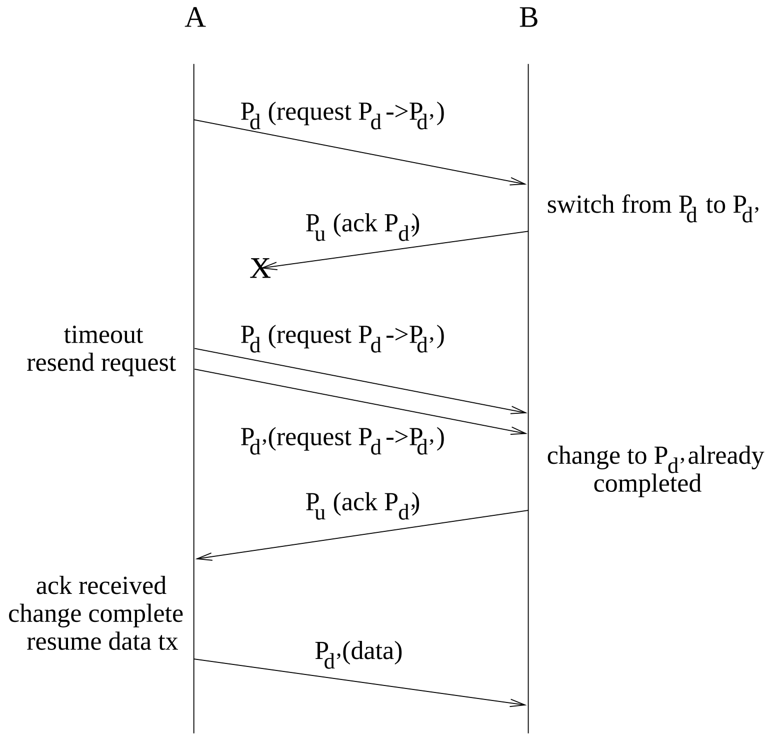

Figure 3-3: Protocol to modify a downstream link, with lost messages and retransmission.

During this message exchange, it is possible that host A may decide to change its request to a third alternative, P_{d*}, before the modification is complete. This may happen if conditions on $P_{d'}$ are rapidly deteriorating and host A decides to bypass that channel. In terms of the protocol, this occurs when host A decides to issue a new request before it receives an acknowledgement for the old request. In this case, the same issue as above of host B's state applies; host B could be listening on either P_d or $P_{d'}$. Thus, this new request to change to P_{d*} must be transmitted over both P_d and $P_{d'}$. Also, in the event host A does not receive an acknowledgement of the change to P_{d*}, it does not know on which channel host B is listening. Thus, host A must retransmit the request on *three* channels, P_d, $P_{d'}$, and P_{d*}.

While this extension to three alternatives may be useful, extending to even more alternatives becomes inefficient and unwieldy. If there are n alternatives, retransmissions of requests must be sent over n channels, requiring more time to retransmit the request and more memory to hold the software for all of the physical layer implementations. Thus, the number of possible alternatives should be limited, and if the limit is reached, the current request must be allowed to complete.

3.2.2 Upstream Modification

Now, suppose that the initiating host wishes to modify its upstream link. Because it does not control the data flow through its upstream link, a similar message exchange to the above cannot be used. This is because a response may come over the new link, or data may still be transmitted over the old link.

A possible solution is for the initiating host to listen continuously for a response over both links (the opposite of the first case). This would work, but it forces a host to run two receivers *simultaneously* since the data may come at any time. This is computationally expensive. A better solution is to force the downstream host to request a change in its downstream link. In this case, a host sends messages on two links *serially* since it has control over the data on its downstream link.

The protocol to execute a change in the upstream link is shown in figure 3-4. Host A, which is initiating the change, sends a message asking host B to request a modification in host A's upstream link. Once host B receives this message and replies by beginning the basic protocol, the same message exchange as in the previous case works.

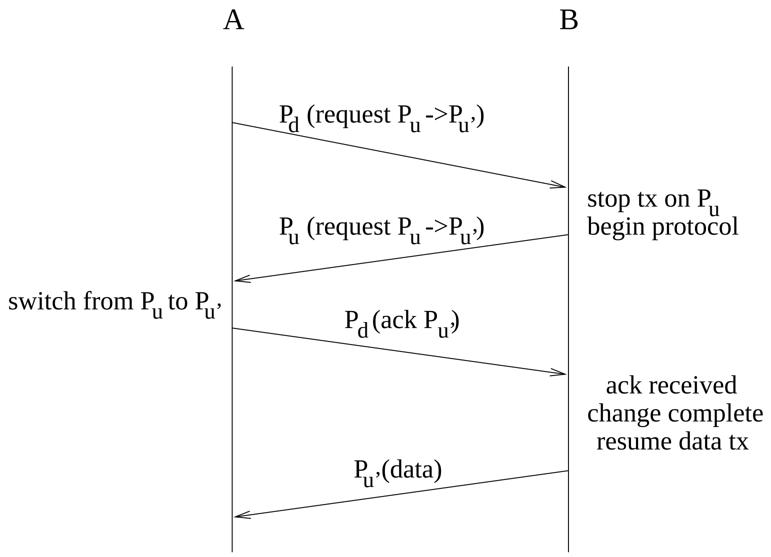

Figure 3-4: Basic protocol to modify an upstream link.

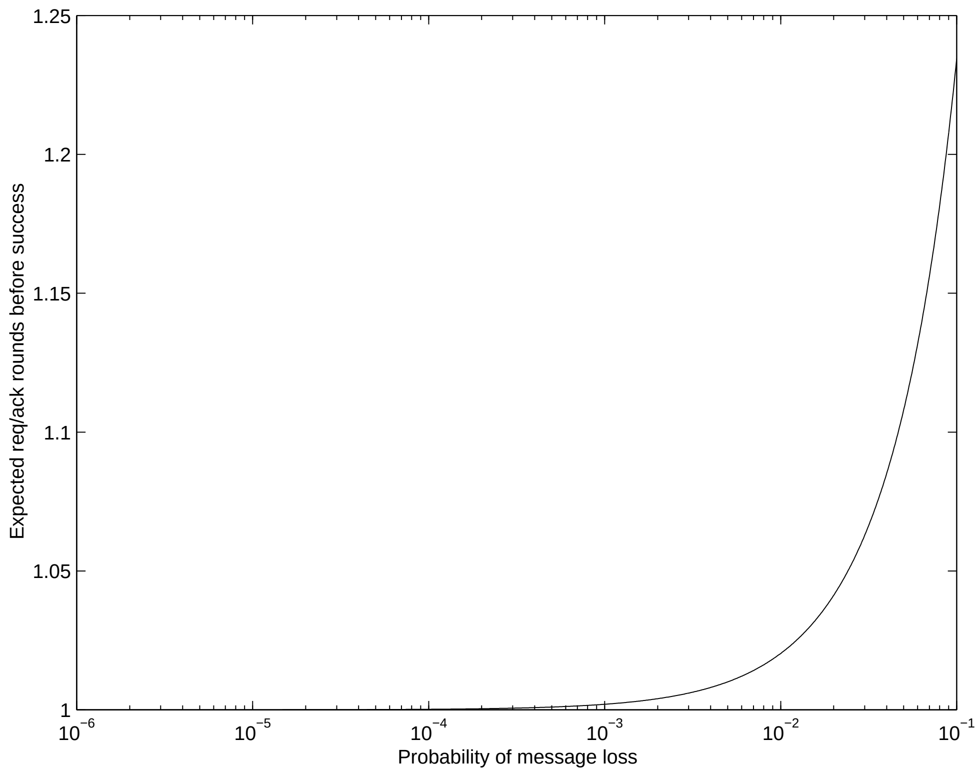

Figure 3-5: Expected number of rounds until success as a function of message loss probability.

3.3 Convergence

This protocol performs a modification reliably in the presence of message losses by retransmitting requests. Although it is possible that the protocol never completes because of message losses, this section shows that the protocol converges and is successful very quickly under a wide range of conditions.

Assuming that the upstream and downstream links are symmetric, each message has an independent probability p of being lost, and the probability that both a request and its acknowledgement are received successfully is $q = (1-p)^2$. This protocol is successful when the first request/acknowledgement pair is successfully communicated. Thus, the probability distribution of number of request/ack pairs that are transmitted before convergence is a geometric distribution with parameter q, i.e.:

$$Pr[success\ in\ n\ req/ack\ rounds] = (1-q)^{n-1} \cdot q$$

From the properties of the geometric distribution:

$$E[req/ack\ rounds\ until\ success] \;=\; \frac{1}{q} = \frac{1}{(1-p)^2}$$

$$
\begin{aligned}
Pr[success\ within\ n\ rounds] &= 1-(1-q)^n \\
&= 1-(2p-p^2)^n
\end{aligned}
$$

With each request transmitted, the probability of failure drops exponentially. Figure 3-5 shows the expected number of rounds until the protocol succeeds as a function of the probability of message loss. Figure 3-6 shows the probability of successfully executing the protocol within two attempts. With a probability of message loss of $p = .01$, the expected number of rounds until completion is 1.02 and the probability of success within two attempts is 99.96%

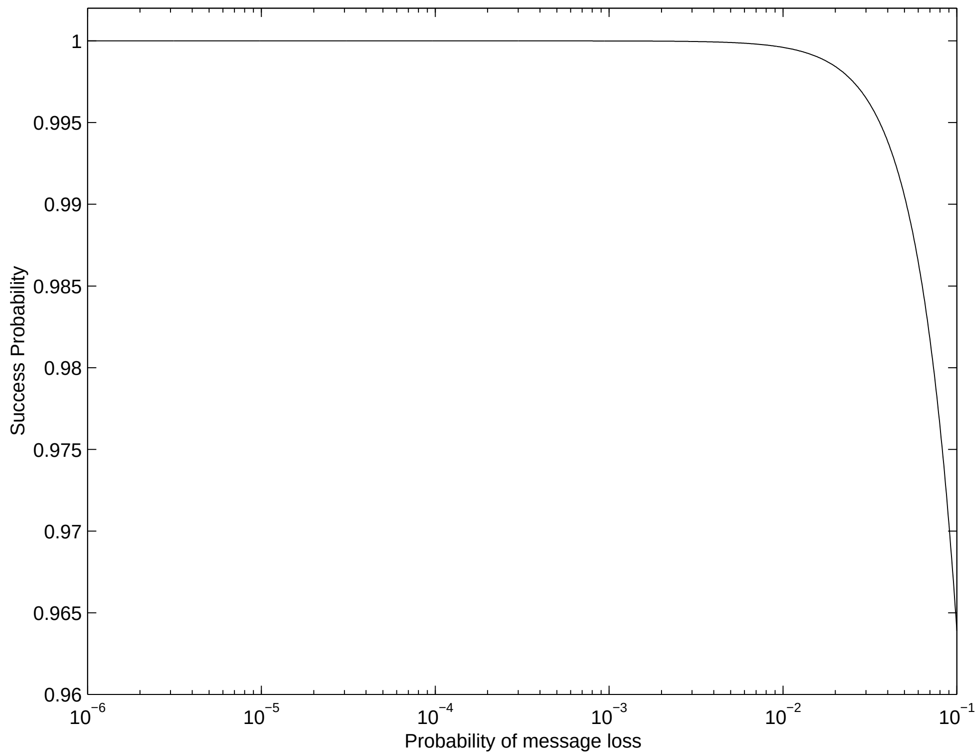

Figure 3-6: Success probability after two attempts of the protocol as a function of message loss probability.

3.4 Simultaneous Request Resolution

To the protocol, the design of a particular implementation of an adaptive wireless network is unknown. Thus, to the protocol, the source of modification requests can be arbitrary. For example, in a network with a single basestation and multiple mobiles, the basestation could be the sole initiator of modifications, but in a decentralized network of mobiles, each individual host may request its own changes. In the second example, either host on the ends of a link may request a modification at any time, and it is possible that both may request changes at the same time. However, only one change can be executed at a time, so there must be a provision in the protocol to resolve any simultaneous request (collision).

A collision is detected when a host that is listening for an acknowledgement to its request instead receives a request from the other host. If a collision is detected at one end of the wireless link, it is likely that a collision was also detected at the other end. Since all requests are treated equally, there is no provision for one host to defer to another in the event of a collision, and each host will attempt to get its request serviced. Thus, to avoid "livelock", where each host retransmits, resulting in another collision and repeating the process, the protocol introduces randomization. When a collision is detected, the host ignores the incoming request, cancels its previous outgoing request and waits a random length of time before retransmitting.

The remainder of this section focuses on the two node case. When extending to three or more nodes, the solution is the same; since only one node can modify the physical layer at a time, in the event of a collision, a randomization mechanism is used to determine which node is allowed to proceed. Since the analysis of such situations is complicated, the two node case is analyzed in-depth to provide insight on the issues involved in this type of randomized solution.

There are two cases to consider: both hosts receive requests and notice the collision, or only one host notices the collision. When both hosts notice the collision, both will ignore the request, and each host will wait a random amount of time before retransmitting its request. Since each waiting time is random, it is likely that one host will retransmit its request before the other host.

When only one host notices the collision, the two hosts are asymmetric. Suppose that host B notices the collision. Host B is waiting a random amount of time before retransmission, while host A is awaiting an acknowledgement. Host A will eventually timeout and retransmit, but the deterministic nature of the duration of the timeout is undesirable. If host B waits the random amount of time and retransmits before host A times out and retransmits, host A receives the request, notices a collision, ignores the request, and begins waiting a random amount of time. Host B is now awaiting an acknowledgement, and the situation has not improved.

In order to ensure randomness at both hosts, after a timeout, the protocol waits a random amount of time before retransmitting a request. Thus, detecting a collision is effectively an immediate timeout. This simplifies the protocol because collisions do not have to be handled by a special case.

The interval from which the random wait duration is chosen is an important parameter. A larger interval reduces the probability that the retransmissions collide but increases the amount of time required to execute the protocol. A collision of retransmissions occurs when both hosts retransmit within a period of time equal to the delay between the transmission time and the reception time. This delay is half of the round-trip time from one host to the other host and back again.

3.4.1 Simplified Analysis

To determine the optimal interval from which the random wait duration is chosen, let us determine the probability that the retransmissions collide, given that no losses of retransmissions occur. Assume each host begins its random wait at the same time. This is the worst case, because if one host begins earlier, the overlap of the random wait intervals is smaller and the probability of a collision is smaller.

This random wait interval is also used after timeouts due to lost messages. Thus, the width of the interval affects the time between retransmissions, which is equal to the timeout plus a random number chosen from the interval. In order to decouple the determination of the optimal random wait interval for collisions and the determination of the optimal time between retransmissions, we can determine the optimal random wait interval independently and then adjust the timeout such that the expected value of the sum of the timeout and the random number is optimal.

Let D be the delay between the transmission and reception times, and let r_1 and r_2 be the random wait durations for the two hosts that are chosen uniformly from 0 to R. In practice, D is not constant, so this analysis uses D to represent the expected value of the delay. A collision occurs when $|r_1 - r_2| < D$.

For a given r_1:

$$Pr[|r_1 - r_2| < D] = \begin{cases} \frac{r_1+D}{R} & \text{if } 0 < r_1 < D \\ \frac{2 \cdot D}{R} & \text{if } D < r_1 < R - D \\ \frac{R-r_1+D}{R} & \text{if } R - D < r_1 < R \end{cases}$$

Evaluating over all values of r_1, we get:

$$Pr[collision\ of\ retransmissions] = P_C = \frac{2 \cdot D}{R} - \frac{D^2}{R^2}$$

We want to minimize the expected amount of time spent in resolving a collision. Let us use the following simplified model for collision resolution that does not take into account the potential differences in waiting time between successes and failures:

- With probability $1 - P_C$, a random wait time expires, a delay of D is incurred, no collision occurs and the change is completed.
- With probability P_C, a random wait time expires, a delay of D is incurred, a collision occurs, and another attempt is made at resolving the collision.

If T_{CR} is the time to resolve a collision, the following recurrence describes T_{CR}:

$$\begin{aligned} T_{CR} &= random\ wait + D + P_C \cdot T_{CR} \\ T_{CR} &= \frac{random\ wait + D}{1 - P_C} \end{aligned}$$

(3.1)

Thus, the expected time to resolve a collision, $E[T_{CR}]$, is:

$$E[T_{CR}] = \frac{E[random\ wait] + D}{1 - P_C}$$
$$E[T_{CR}] = \frac{R/2 + D}{1 - \frac{2 \cdot D}{R} + \frac{D^2}{R^2}} \quad (3.2)$$

Figure 3-7 is the graph of equation 3.2. When the range R is very small, the expected T_{CR} is large since the probability of a collision is very high. When R is large, the probability of a collision is low, but the expected T_{CR} is large because the random wait takes a long time to complete. The minimum expected time to resolve a collision occurs when $R = 4D$.

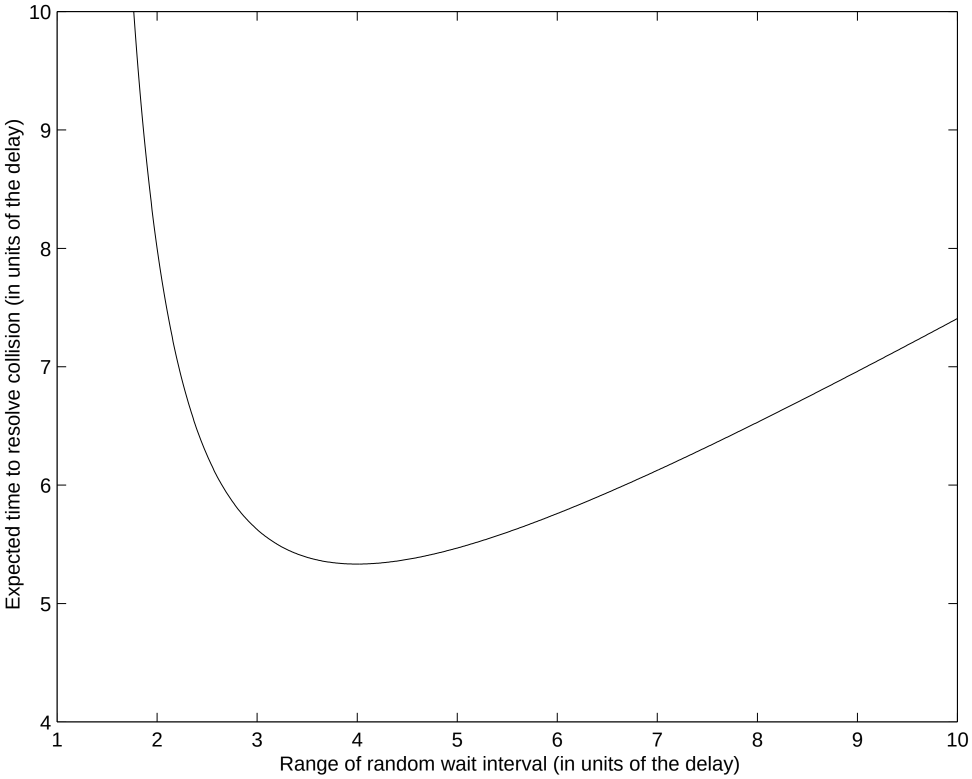

Figure 3-7: Expected time to resolve a collision as a function of the range of random wait times. All times in units of D.

3.4.2 Simulation

The previous section left out several factors which facilitated a straightforward analysis of the resolution of collisions. In particular, the analysis above did not take into account message losses or the potential offset in the times at which each host begins its random wait. Also, this analysis assumed that the expected waiting time per attempt was constant, regardless of whether the attempt succeeded or resulted in another collision. This simplified analysis resulted in a rough estimate of the optimal value of R and also provided insight into the factors that affect the amount of time required to resolve a collision. However, in order to obtain a more realistic optimal value of R, the three factors that are mentioned above were incorporated into a simulation of the collision resolution.

One trial of this simulation involved setting up the initial collision, selecting random wait times, and determining if either of the retransmissions were lost. If so, or if the retransmissions resulted in another collision, the process of selecting random wait times repeats until no collision is detected. The time to resolve a collision is defined to be the amount of time elapsed from the moment the first host receives a conflicting request to the moment a host receives the first non-conflicting request, which results in an acknowledgement.

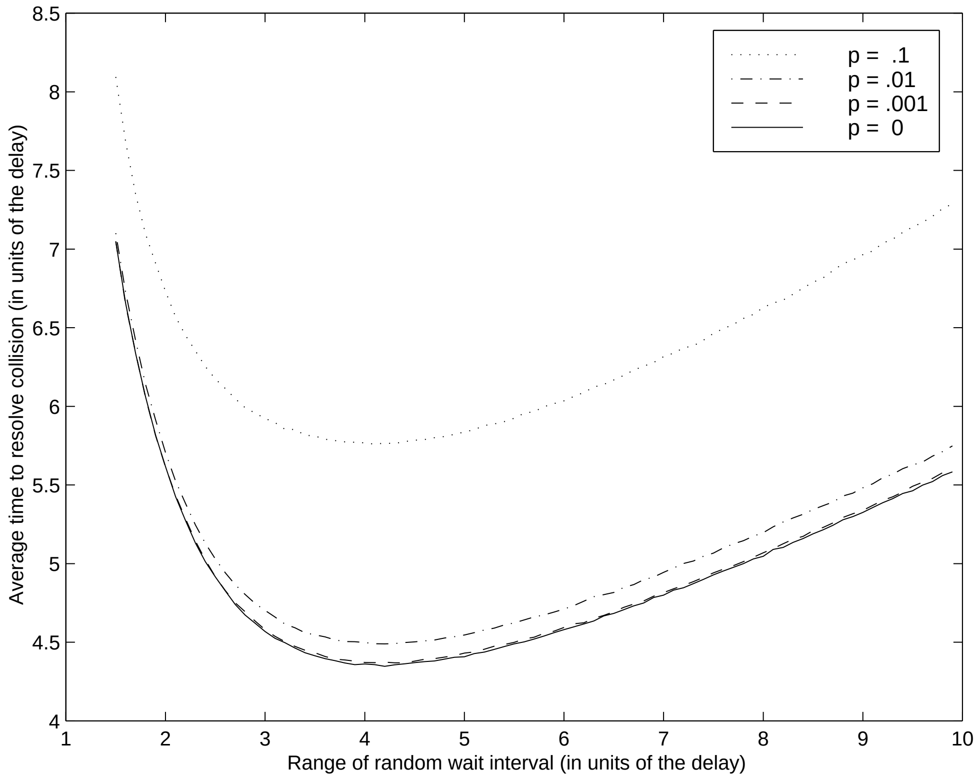

Figure 3-8: Average time to resolve a collision as a function of the range of random wait times with different message loss probabilities. All times in units of D.

Figure 3-8 is the average time to resolve a collision reported by the simulation over a range of R and p values, where p is the probability of a message loss. For each R value in increments of $.1D$ from $1.5D$ to $10D$, four different p values were used ($p = 0, 10^{-3}, 10^{-2}, 10^{-1}$), and for each p value, the simulation was run for 1 million trials. Also, the timeout was set to $3D$ [1].

The shape of the curves produced by the simulation is the same as the one derived from the analysis in figure 3-7, and the average time to resolve a collision is minimized at about the same R value ($R = 4.2D$). This suggests that the additional parameters factored into the simulation do not greatly affect the optimal R. Since the value of R determines the probability of collision of two messages that are *not lost* and the resulting latency penalty, the probability of message loss does not affect the optimal R. Also, the potential offset in the times at which each host begins its random wait is small compared to R, and thus it does not affect the optimal R that much. Finally, the difference in the expected waiting time between collisions and successes is only observed on the very last attempt, since that is the only one that succeeds. Thus, it has no effect on the optimal R. However, it does affect the time to resolve a collision. The protocol succeeds when one of the two hosts transmits D time before the other. The expected waiting time on this attempt is the smaller of the two waiting times, not the expected time of a single random wait, as was used in the simplified analysis. This distinction accounts for much of the difference in average waiting time between figures 3-7 and 3-8.

3.5 Summary

Since the physical layer of a wireless network link is completely implemented in software, modifications can be made by simply changing the code. However, both ends of the link must quickly switch to the same new physical layer to minimize the latency incurred by the modification. Messages must be passed between the two ends to identify the new physical layer (in terms of the software radio layering model), and to synchronize the modification. Also, since these control messages may be lost and multiple modification requests may be active at one time, a protocol must be used to ensure reliable modifications to the physical layer of the wireless link.

The protocol described in this section meets the desired criteria. In most cases, a simple message exchange involving one request and one acknowledgement is all that is needed to execute a modification. In an unfavorable environment, a retransmission scheme allows the modification to occur quickly in the presence of message losses. Also, since the mechanism to decide when modifications are necessary is unknown to the physical layer and multiple modification requests may be active, randomness is built into the system to ensure that the protocol completes the desired changes without deadlocking.

[1] The timeout is the amount of time, after transmitting a request, a host waits before retransmitting its request. The minimum amount of time it takes for an acknowledgement to reach the host is $2D + C$, where D is the delay from one host to the other, and C is the time required to change the physical layer code. Using the approximation $C \approx D$, the acknowledgement is expected in $3D$, so if it is not received by this time, retransmit the request.

Chapter 4

System Design and Implementation

Traditionally, a network interface device for both wired and wireless LANs consists of the physical device, such as an ethernet card or a WaveLAN device, and its device driver. The model of this type of architecture is shown in figure 4-1 (a). A user application accesses the network by interfacing with the top of the network stack. Information is processed down the network stack, and packets are communicated through the device driver and the hardware device to the network.

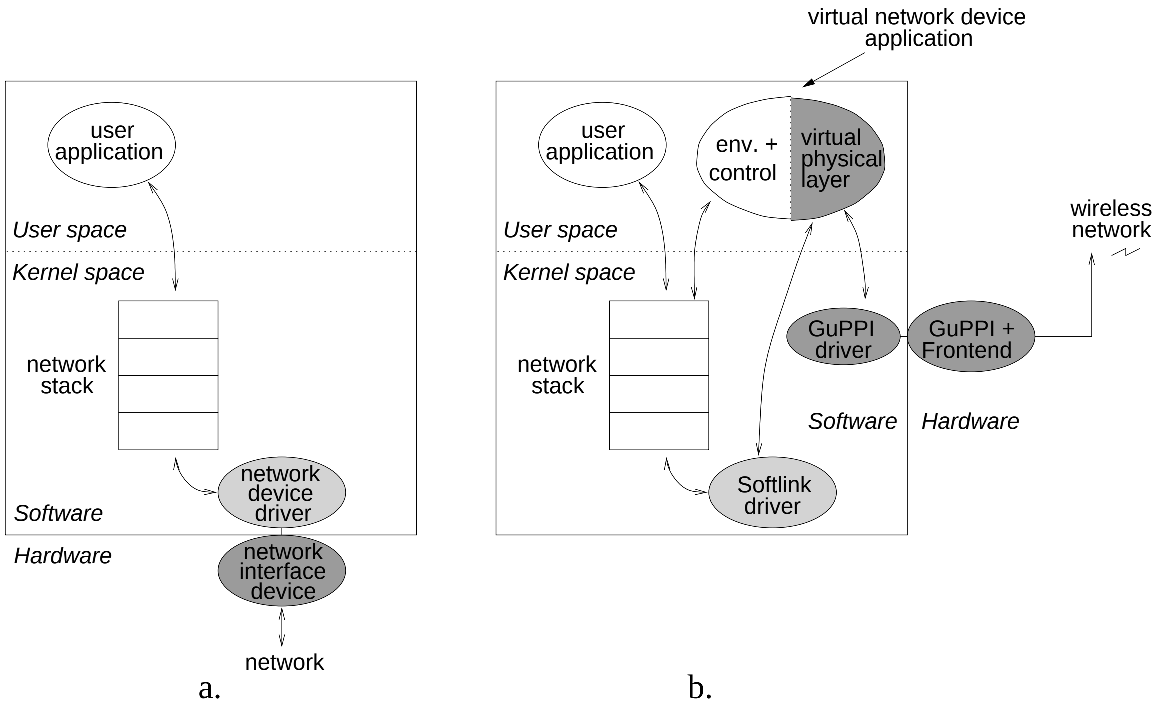

Figure 4-1: Comparison of the (a) traditional approach with the (b) SpectrumWare approach.

Our approach, also shown in figure 4-1 (b), takes many of the functions provided by the hardware network interface device of the traditional approach and performs them in software. User applications still access the network by interfacing with the top of the network stack, and a device driver still communicates with the bottom of the network stack. However, this device driver, instead of communicating with a piece of hardware, communicates with the software application that implements the physical layer. The processing that is normally performed on the network card is now done in software, and the interface to the physical world is provided by the GuPPI and the frontend. By implementing the physical layer in software, the functionality of the software-based network interface can be modified without changing the hardware.

This chapter discusses the design of the software-based adaptive wireless network and its implementation in the SpectrumWare architecture. The scope of this design is limited to the infrastructure that executes a desired modification. The determination of which modifications are necessary is not addressed in this thesis but is discussed as future work in chapter 6.

4.1 Virtual Network Device Application

The SpectrumWare virtual radio system runs on any processor with a PCI bus running Linux. Applications, such as a wireless network device, reside in user space, and SpectrumWare extensions to the operating system allow access to system resources such as the network stack and physical devices.

The SpectrumWare wireless network device, as shown in grey in part (b) of figure 4-1, consists of three main components:

- **Softlink device.** The Softlink device appears to the operating system as a network device and moves packets between the network stack and the virtual physical layer, which resides in user space.

- **SPECtRA application.** The SPECtRA application performs all of the physical layer processing that is traditionally performed on a network interface card. It also provides the functionality to execute modifications to the physical layer.

- **GuPPI.** The GuPPI, as described in section 2.1.1, transports the waveform samples generated by the application to the analog front end.

The virtual network device application is implemented in user space as a SPECtRA application. Thus, its structure is comprised with two partitions: the code modules that implement the virtual physical layer and the programming environment that controls the data flow through the modules. The following subsections describe each of these aspects.

4.1.1 Virtual Physical Layer

The physical layer exists as a chain of processing modules that implements all of the functions that are required to transform packets into waveforms and vice versa. The virtual physical layer, shown in figure 4-2, consists of two processing chains, one for transmit and one for receive. Just like any other SPECtRA application, each processing chain begins with a source and ends with a sink [2]. To transmit, the Softlink Source, which is an interface to the Softlink device driver, takes a queued packet from the device driver and moves it into the physical layer application. After the data bits are extracted from the packets, the remaining modules perform the physical layer functions (MAC, channel coding, line coding, modulation, and multiple access). The output of the last physical layer module is then passed to the GuPPI Sink, which takes the samples and transfers them to the GuPPI for transmission. The receive chain works in a similar manner, but in reverse. This chain begins by taking samples from the GuPPI and ending with packets that are sent to the Softlink device driver.

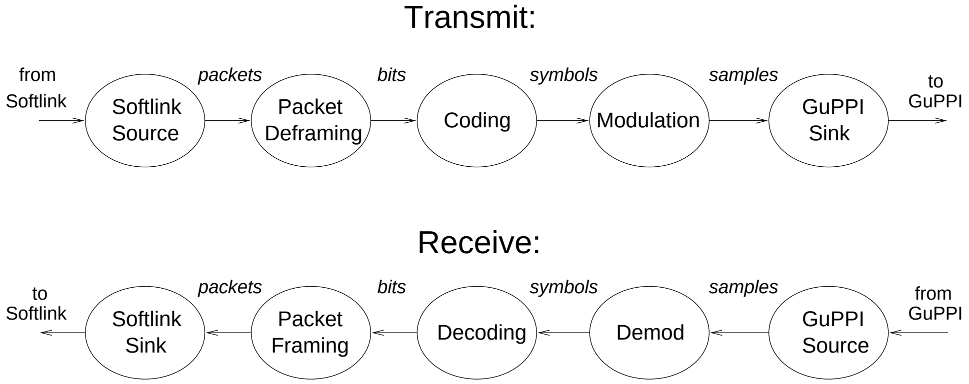

Figure 4-2: The Virtual Physical Layer performs the processing that is traditionally performed on the network interface card.

There is not necessarily a one-to-one mapping of modules to layers of the software radio layering model. That is, a module may perform partial processing for one layer, or a module may combine the functionality of two layers. For example, a filter module performs only part of the processing for the modulation sublayer, with the remainder of the processing performed by the demodulation module. In this case, the separation of functionality of one layer into two modules makes sense, since the choice of filter and demodulation technique are fairly independent.

Combining layers into one module may be necessary for computational efficiency [2]. For example, consider a transmit application with a data rate of 100 Kbits/sec that uses frequency hopping for multiple access and has a sampling frequency of 10 MHz. One possible implementation is a two step process that modulates at baseband (modulation function) and then translates that signal up to the current IF frequency specified by the hopping sequence (multiple access function). The translation up to an IF involves multiplying at the output sampling frequency, which is very computationally intensive. A more efficient implementation is to combine the modulation and multiple access modules and directly modulate to the current hop frequency, which eliminates the multiplications at the sampling rate. In this example, the computationally more efficient implementation saves 100 complex multiplications for each transmitted bit.

4.1.2 Control Band

The out-of-band control portion of the virtual network device application, shown in figure 4-3, is responsible for all of the operations on the physical layer outside of the actual data processing [2]. At minimum, the control script creates the modules for the physical layer, connects them together to construct the topology of the physical layer, and initializes the processing chain. This set of actions creates a physical layer and runs it, as is, until the process is terminated.

To make dynamic adaptation possible, the control band performs many other important operations. The control band:

- Communicates with other processes or other hosts to receive information on modifications that are to be performed.
- Implements the protocol from chapter 3 to coordinate physical layer modifications.
- Modifies the physical layer by changing the parameters or creating and destroying processing modules.

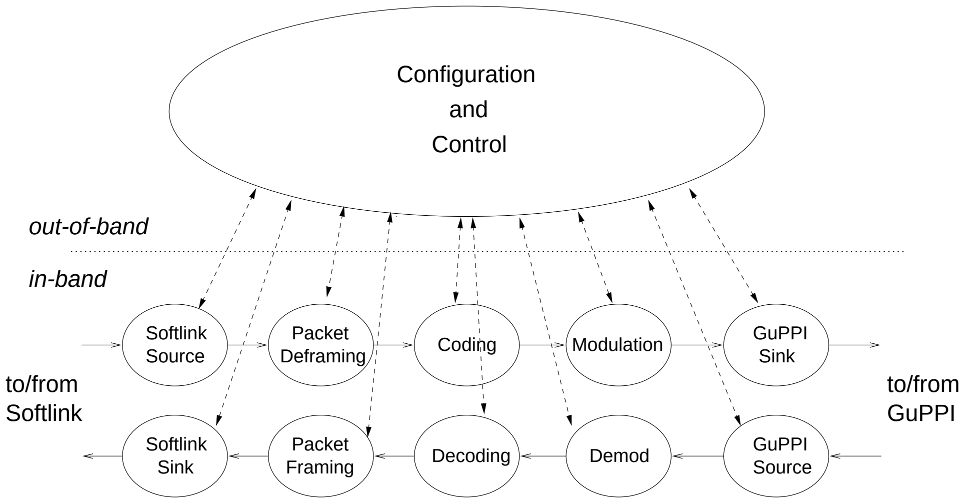

Figure 4-3: The structure of an example Virtual Network Device.

4.2 Executing Physical Layer Modifications

The SPECtRA environment provides a platform upon which a virtual network device can run. In order to use the virtual network device to execute dynamic modifications to the physical layer, additional functionality is needed. This section discusses the implementation of the adaptive capability of the virtual network device, including the passing of messages to and from the physical layer and the dynamic linking of modules.

4.2.1 Controlling the Adaptive Physical Layer

There are two main configurations for wireless networks: a single basestation with multiple mobiles or a network of peers. In the single basestation model, the basestation has sole control over the physical layer of the network, so only the basestation can initiate modification requests. This model is fairly simple, since no conflict can arise while attempting to modify the physical layer. In a peer network model, control over the physical layer is distributed throughout the network; any host may initiate modifications to the physical layer. Since there are many potential sources of modification requests, this model is more complicated. This network configuration is the focus of much of this thesis, especially in chapter 3.

The virtual network device makes up half of the operation of the adaptive network. Since the virtual network device application is responsible for only the execution of a modification, it must receive information from another source as to what modification to implement. This source is the intelligent network management application that serves as the intermediary between user applications and the virtual network device. This model of operation, shown in figure 4-4, provides a clean interface where the user applications supply their requirements, the virtual network device reports the current channel conditions, and the network management application incorporates all of this information and determines what changes to the physical layer are necessary and when they should be implemented. As long as a particular host is allowed to initiate modifications to the physical layer, these orders to change the physical layer are passed to the virtual network device for execution.

From the perspective of the virtual network device, a modification request may originate from two sources: the network management application running on the local host, or a neighboring host. The first source represents a modification request originating from the local host, while the second source is a request originating from another host on the network. These requests must be communicated in order to implement the required modifications.

Information must also flow from the physical layer to other processes or hosts. For example, bit error rates can be measured in the physical layer, and current information about the condition of the wireless link may be useful to the network management process that is measuring communication quality and attempting to optimize over all

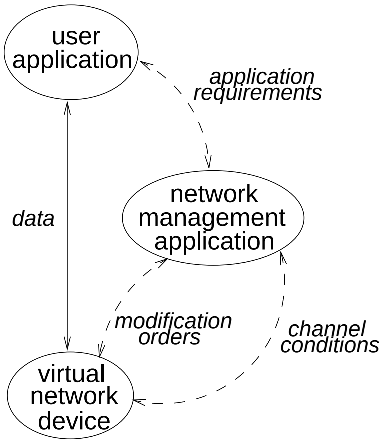

Figure 4-4: The interface of the network management application.

(network, link, and physical) layers. In order to support this information flow, there must be a mechanism for communication between processes and hosts.

The virtual network device application, as a SPECtRA program, is just another user application from a computing standpoint. As such, it can exploit existing support in the Linux operating system for interprocess and network communication. Specifically, the control band of the SPECtRA application can open a socket connection to either another process or to the network stack for network traffic, just like any other user application. Through these socket connections, the control band receives input with the description of the new physical layer and initiates the modification protocol to execute the change.

4.2.2 Modular Program Structure

An effective virtual network device must be able to implement any possible physical layer formed by any legal combination of processing modules. This includes new modules that have not yet been invented or implemented. For example, if a new coding technique is invented, the new technique can be written as a module, compiled, and added to the module library. There should be little, if any, alterations required to the control band portion of the virtual network device.

The current practice for programming SPECtRA applications does not support this type of flexibility. The control band of a SPECtRA application is currently written as a script, where all of the needed modules are coded into the script [2]. While this allows a certain degree of flexibility by allowing changes between a pre-defined set of modules, it does not allow the easy introduction of new modules. The control band must be rewritten whenever new modules are added.

In order for the control band to support this transparent upgradability, a new programming practice is needed, where the module libraries are dynamically linked to the control band. In this model, the control band does not have any references to specific modules. Instead, whenever the control band needs to create a module, it loads the correct dynamic library, determines the correct pointer to the module constructor from the symbol table, and creates the module by executing the constructor. As long as the library and symbol names corresponding to the desired module are passed to the control band when requesting a modification, processing modules for the physical layer can be created in this manner.

There are two issues with creating modules in this manner. The first is that the control band does not have complete information on the details of the processing module it wants to create. That is, the specific type of the module is unknown, and type information is needed to allocate memory for a module and call its constructor. To handle this issue, the control band uses the hierarchy of C++ classes in the SPECtRA environment. All processing modules, which are implemented as C++ classes, are subclasses of class `VrSigProc` [2]. This class contains the methods common to all processing modules, including a constructor. Even though the control band treats the constructor of the dynamically linked module as a constructor for class `VrSigProc`, calling it executes the desired module's constructor and creates the correct module. Thus, modules can be created without being explicitly specified in the control band.

Since modules are created dynamically, they cannot be identified by name in the control band. A naming convention needs to be developed so that modules can be identified and located in the module library. In this convention, instead of calling a module by name (eg. `VrFHFSKMod`, or Frequency Hopping FSK Modulator), a module is identified by the file in which it resides (eg. `libspectra.so`), and the symbol name that corresponds the the module's constructor (eg. `__t10VrFHFSKMod1ZciPPc`).

The second issue with creating modules dynamically is that different modules have different types and numbers of operating parameters. Since a dynamically created module must be called with a constructor that resides in the base class `VrSigProc`, there must be a general convention for passing a variable number of arguments of varying types to the constructor. Our solution is to use a format similar to the way C supports command line arguments [12]. The constructor is called with two arguments: an integer, `argc`, representing the number of parameters being passed, and a pointer to an array of character strings, `argv`, which contains the parameters as strings.

4.2.3 Dynamic Code Loading

Active networks use a scheme to dynamically introduce new services to network nodes that were not anticipated when the networks were originally designed [19]. A similar scheme is used here to dynamically load code into existing hosts corresponding to new physical layer implementations. The modular structure, described in section 4.2.2, lends itself to the dynamic loading of new code into the module library.

Ideally, the virtual network device periodically contacts a central code repository through its wireless network connection and checks to see that its module library is up to date. New or revised modules are downloaded thorough the wireless network and saved locally. In this situation, code upgrades occur transparently to the user, and each virtual network device has a copy of all of the processing modules.

However, there may be situations where a host does not have a complete set of modules. For example, this may occur if the virtual network device has not performed an automated upgrade for a long period of time, or if storage constraints on the host prevent local storage of the entire module library. In this case, dynamic loading of code is possible on a demand basis. The modification protocol from chapter 3 is extended so that if a modification request requires a module the host does not have, the host can download that particular module.

This ability to "learn" new functions on the fly is a huge advantage of the software radio approach. All possible physical layer instantiations need not be present when the software radio network device is installed on a host, and new standards or upgrades need not be manually installed. New code can be loaded onto a host automatically and transparently to the user.

Chapter 5

System Performance

There are two aspects to performance for the SPECtRA-based adaptive network. The first metric is the performance of the software physical layer in the absence of adaptations. These measures show the overhead, throughput, and latency of our software approach for physical layer processing. The second performance measure deals with the adaptations. Whenever a wireless link is modified, data transmission must stop for a period of time. This latency is incurred under the assumption that the new wireless link will provide better performance. This chapter describes the performance of the adaptive physical layers in these two contexts.

5.1 Software Physical Layer Performance

There are three measures of performance for the software physical layer:

- **Overhead**, the percentage of the CPU that is required to operate the software physical layer.
- **Throughput**, the maximum possible rate of data transfer.
- **Latency**, the processing delay of a packet through the physical layer.

The first two measures, overhead and throughput, are closely related. Each symbol that passes through the software physical layer requires a certain amount of processing time. As the symbol rate increases, the amount of CPU processing per unit time, which is the overhead, increases. The maximum possible throughput is the rate that results in an overhead of 100%. Using a Pentium II 450 and 1 bit per symbol, the maximum possible throughput is 125 kpbs. Table 5.1 shows the overhead for various data rates.

The third measure, latency, is a large factor in the overall performance of the wireless network. If the amount of processing time required by the software physical layer is high, there is a larger delay between the time a packet is sent from one host and

Data Rate	CPU overhead
100 kbps	80%
50 kbps	33%
10 kbps	<1%

Table 5.1: CPU overhead of the software physical layer running on a Pentium II 450 and using one bit per symbol.

received at another. The remainder of this section breaks down the components of the latency and identifies the bottlenecks in the processing.

5.1.1 Latency Components

The path of a piece of data traveling through the SPECtRA-based adaptive network can be broken down into several parts. The components of this path, shown in figure 5-1, are the network stack, Softlink device driver, SPECtRA physical layer, GuPPI, the air, and the corresponding components on the destination host. In comparison, the path for data on a wireless ethernet network, such as RadioLAN, also shown in figure 5-1, consists of the network stack, network device driver, wireless ethernet card, and the air.

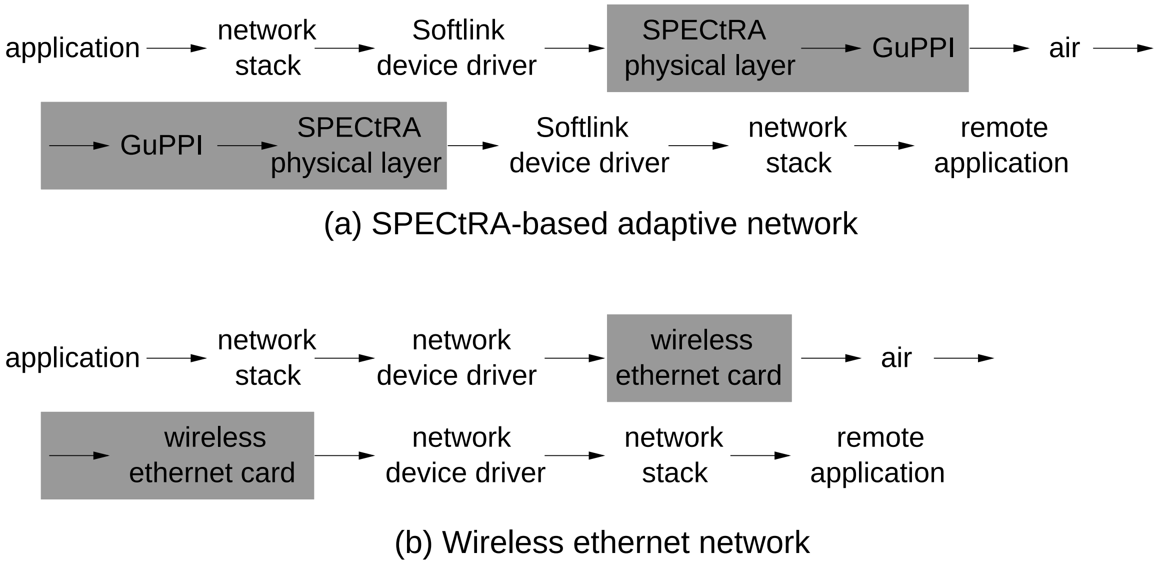

Figure 5-1: Path through the components of a wireless communication link for both (a) a SPECtRA-based network and (b) an ethernet network.

The total delay for a piece of data through either of the two communication systems is the sum of the delays through each of the components. The adaptive network path and the ethernet path share many common components, namely the application, network stack, device driver, and the air. Thus, the difference in the total delay of the two systems is caused by the shaded components of figure 5-1. The wireless network card in the ethernet network are replaced by the SPECtRA physical layer and the GuPPI.

The wireless ethernet network characterized here is a RadioLAN network with a raw transmit data rate of 10 Mbps. For this system, the propagation delay for small packets through the air is insignificant. For example, a 32-byte ping packet requires .026 msec of transmission time. Thus, assuming the delay through the ethernet card is also extremely low, the delay through the wireless ethernet network is mostly due to the unshaded components (network stack and device driver) of figure 5-1.

For the SPECtRA-based network, the total delay is given by the sum of the delays through the unshaded components, SPECtRA physical layer, and the GuPPI. Since the delay through the unshaded components is approximately the total delay through the wired network, the total delay through the SPECtRA-based network, not counting the propagation delay through the air, $T_{SPECtRA}$, is:

$$T_{SPECtRA} \approx T_{ethernet} + T_{TX} + T_{GuPPI\ tx} + T_{GuPPI\ rx} + T_{RX}$$

where:
$T_{ethernet}$: total delay through the wired network
T_{TX} and T_{RX}: delay through the software physical layers for transmit and receive
$T_{GuPPI\ tx}$ and $T_{GuPPI\ rx}$: delay through the GuPPI for transmit and receive

5.1.2 Results

The UNIX ping application is ideal for measuring the total delay of a network since it sends fixed size packets and returns the round trip time. Using a RadioLAN wireless ethernet network, the average round trip time from a machine to its wireless gateway is 1 msec. Since the round trip time is composed of two transmissions (request and reply), the average one way delay, $T_{ethernet} = .5$ msec.

To measure the performance of the SPECtRA-based network, an asymmetric connection is set up between the two dual Pentium II 450 computers. That is, the network connection from host A to host B consists of a SPECtRA-based wireless link, while the return path from host B to host A is a 100 Mbit wired ethernet link[1]. Using the ping application, the average round trip time for a system with a data rate of 100 kbps is 28.4 msec. Since the return path is a wired ethernet link with a one way delay of .05 msec, the average one way delay of the wireless link is 28.35 msec.

The propagation delay through the air is determined by taking the number of symbols transmitted and dividing by the symbol rate. An 84 byte ping packet transmitted

[1]Current hardware issues prevent the construction of a two-way wireless link. The GuPPI is currently unable to simultaneously support transmit and receive functionality.

by this SPECtRA-based network, using one bit per symbol, is about 700 symbols in length, and thus, with a rate of 100 Ksymbols/sec, the propagation delay is 7 msec. Thus, 7 msec is subtracted from the average one way delay of 28.35 to obtain $T_{SPECtRA} = 21.35msec$. Table 5.2 gives the breakdown of the individual component delays for the ping application sending 84 byte packets [2].

Component	**Average Delay**
$T_{SPECtRA}$	21.35 msec
$T_{ethernet}$	.5 msec
T_{TX}	11.46 msec
$T_{GuPPI\ tx}$	2 msec
$T_{GuPPI\ rx}$	.6 msec
T_{RX}	6.5 msec

Table 5.2: Average delay per 84 byte ping packet through the components of the wireless link.

$T_{GuPPI\ tx}$ and $T_{GuPPI\ rx}$ occur because the GuPPI performs buffering during both transmit and receive operations. While buffering is required for proper operation of the GuPPI, excess buffering increases delay. On the transmit side, a preceding buffer may still be undergoing transmission when the current buffer is created and queued for transmission, and on the receive side, an entire buffer worth of data must be collected before it is handed off to the physical layer for processing. With 10 page buffers[3] and a transmit sampling rate of 10 Mhz, the average delay time through the GuPPI for transmission is:

$$T_{GuPPI\ tx} = \frac{1}{2} \cdot \frac{4096 \cdot 10}{10000000} = 2\ msec$$

Similarly, the average delay time through the GuPPI for reception, with 10 page buffers and a receive sampling rate of 33 Mhz, is:

$$T_{GuPPI\ tx} = \frac{1}{2} \cdot \frac{4096 \cdot 10}{33000000} = .6\ msec$$

T_{TX} and T_{RX} represent the amount of time required to process a packet in the software physical layer. In this implementation, the transmit software involves reception of a packet from the Softlink driver, packet and byte framing, modulation, and transfer to the GuPPI. Table 5.3 breaks down T_{TX} into its components.

The largest contribution to the delay in the transmit software is in polling the Softlink driver when waiting for a packet. Instead of allowing the software physical layer to

[2]In the table, the sum of the component delays do not exactly equal $T_{SPECtRA}$. $T_{SPECtRA}$ is determined by observing the round-trip latency of the ping application, while the component delays are analytically derived or measured by other means.

[3]1 page = 4096 bytes on the Pentium II.

Transmit Component	Average Delay
Polling Softlink	10 msec
Transfer from Softlink	.24 msec
Packet and byte framing	.34 msec
FSK Modulation	.88 msec
Transfer to GuPPI	0.0 msec
T_{TX}	11.46 msec

Table 5.3: Average processing time per 84 byte ping packet for each component of the transmit software.

continuously poll the Softlink driver, which would cause the physical layer to use up all of the available CPU cycles, leaving none for other processes, upon each poll which results in no packet, the physical layer yields the CPU. However, after yielding (by using the `usleep` command), the physical layer does not get the CPU back for 20 msec. Thus, the time interval between polls to the Softlink driver is 20 msec, and packets which arrive during this interval wait for an average of 10 msec before being processed. When the physical layer does not yield the processor, the average round trip time for the entire system is 18.6 msec, which is about 10 msec faster than the 28.4 msec round trip time observed for the system yielding the CPU. However, while yielding the processor for 20 msec increases the average delay by 10 msec, it decreases CPU utilization from 99% to about 2%.

While yielding the processor is necessary to reduce resource usage, the penalty in packet delay is large. Modifying the length of time for which the processor is given up would greatly improve the delay time. If the yield call could be modified to only yield for 10 msec, the average delay would decrease to 5 msec while increasing CPU usage only by an additional .3%. Yielding for 1 msec would decrease the average delay to .5 msec and increase CPU usage to 7.3%. This would reduce the total transmit latency, T_{TX}, to about 2 msec and the total latency through the SPECtRA-based link, $T_{SPECtRA}$, to about 12 msec.

The receive software involves transferring samples from the GuPPI, channel filtering, demodulation, packet and byte de-framing, and sending the resulting packet to the Softlink driver. Table 5.4 breaks down T_{RX} into its components. The largest contributor to the delay in the receive software is the channel filter. The delay in the channel filter is directly related to the size of the filter. This particular implementation uses a FIR filter with 200 taps. This aspect of the physical layer is one that can be easily modified on-the-fly to improve performance. As environmental conditions improve, the channel filter size can be reduced to save computation and reduce the delay introduced by the filter.

Receive Component	Average Delay
Transfer from GuPPI	0.0 msec
Channel filter	4.2 msec
FSK Demodulation	1.2 msec
Packet and byte de-framing	.1 msec
Transfer to Softlink	0.0 msec
T_{RX}	6.5 msec

Table 5.4: Average processing time per 84 byte ping packet for each component of the receive software.

5.1.3 System Bottlenecks

With the ping application, the largest contributor to the delay through the wireless link is caused by yielding the processor while polling the Softlink driver. While the 10 msec average delay introduced by yielding the processor is 47% of the total delay, it can be reduced by not yielding the processor, or modifying the Linux system call that yields the processor so that it returns more quickly.

The most serious bottleneck in the wireless link is in the software processing functions, especially channel filtering, demodulation, and modulation. Each of these software functions is time consuming because they either require substantial computation per bit (in the case of filtering and demodulation) or they compute on an enormous number of samples at or near the sampling rate (modulation). Alleviating this bottleneck would improve all three measures, latency, throughput, and overhead, of the software physical layer.

5.2 Modification Protocol Performance

In order for a modification to the physical layer to be useful, the benefits of the modification must outweigh the costs. The improved performance that is achieved from using the new physical layer must be worth the latency incurred by the modification protocol described in chapter 3 that executes the change. There are many different ways the performance of a wireless network can be measured, but the most obvious metric is bandwidth. The analysis in this section focuses on the performance of the protocol in the context of a physical layer modification that increases the bandwidth (data capacity) of the wireless link.

5.2.1 Performance Criteria

When a modification to the physical layer is initiated, data communication stops. Messages to coordinate the change at both ends of the wireless link are exchanged, and after the change is completed, data communication resumes. This gap in the data communication can be seen as a "handoff" between two physical layers. Thus, the cost of making a modification is the lost data capacity during this handoff latency.

In this analysis, the benefit of changing the physical layer is increased data throughput. Thus, upon completing the modification, more bandwidth is available, and the network begins to make up the lost data capacity. If l is the handoff latency, and $B_{current}$ and B_{new} are the bandwidths of the current and new physical layers, respectively, the break-even point, T_B, which is the amount of time it takes for the network to make up all of the data capacity lost during l, is given by:

$$(B_{new} - B_{current}) \cdot (T_B - l) = B_{current} \cdot l$$

$$T_B = \frac{B_{new} \cdot l}{B_{new} - B_{current}}$$

Let the ratio of the bandwidths of the new physical layer to the old physical layer be $r = \frac{B_{new}}{B_{current}}$:

$$\begin{aligned} T_B &= \frac{r \cdot l}{r - 1} \\ T_B &= l \cdot (1 + \frac{1}{r-1}) \end{aligned} \tag{5.1}$$

After T_B time, the network has made up all of the lost data capacity and is now running ahead. Thus, if the conditions are stable enough such that another physical layer modification is not needed within T_B time, a modification will be successful in increasing the net bandwidth of the communication link.

5.2.2 Results

The performance of the modification protocol is evaluated while executing two different types of physical layer modifications. The first is a minor modification, where no code modules are replaced. All of the physical layer processing functions are the same, except certain parameters, such as data rate, frequency band, or hopping sequence, are changed.

The second type of modification is a major change, where some code modules are replaced. This occurs when the some aspect of the physical layer processing changes, such as modulation or coding. In this case, one or more code modules are taken out, code is swapped in, and the processing chain is reinitialized. This case will require more processing time than the first case, since code is being changed.

To measure the performance of the modification protocol under these two cases, the same asymmetric connection from section 5.1.2 is set up between two dual Pentium II 450 computers, with one SPECtRA-based wireless link and a return path through the 100 Mbit wired ethernet link. A modification is then executed on the wireless link while keeping the wired link stable.

To measure the performance of a minor modification, the physical layer of the wireless link is implemented as a 100 kbps frequency hopping frequency shift keying (FSK) system. The modification then involves changing the hopping sequence of the wireless link. This requires no change in the physical layer software, since the new hopping sequence is communicated to both ends of the link and passed into the transmitter and receiver as a parameter change. This modification incurs an average latency of 46 msec. Of the 46 msec, approximately 24.9 msec is due to communicating packets to coordinate the modification (24.85 msec on the wireless link, including approximately 3.5 msec in the air, and .05 msec on the wired link). Thus, it takes approximately 21.1 msec to modify the parameters on both ends of the link.

To measure the performance of a major modification, a change in the modulation format is executed. The physical layer is initially implemented as a 100 kbps FSK system, and the modification involves changing the physical layer to a 100 kbps frequency hopping FSK system. This requires that the modulation and demodulation code modules be replaced in the transmitter and receiver, respectively. This modification incurs an average latency of 70 msec. As expected, this modification requires more time since more work must be done to execute the change. In this particular situation, the code for the new modulation format is already resident in memory when the change is requested. If the desired code modules are instead stored on disk, the latency for a modification would be much higher.

The costs of performing physical layer modifications are the latencies described above. In order to determine the benefit of these modifications, the break-even time, T_B is needed. Figure 5-2 is the graph of equation 5.1 for both values of the handoff latency. As the improvement in bandwidth increases, T_B drops off fairly rapidly. Table 5.5 summarizes the break-even times for the two types of modifications at various values of the bandwidth increase. For relatively low bandwidth increases, such as 10%, the break-even time is a fairly small .77 seconds for the major modification. For larger increases in bandwidth, such as 100%, after .14 seconds, the network has made up all of the lost capacity and is now running faster.

Type of modification	**Handoff Latency**	**Bandwidth Increase**			
		10%	**20%**	**50%**	**100%**
Hopping Sequence	.046 s	.506 s	.276 s	.138 s	.092 s
Modulation	.070 s	.770 s	.420 s	.210 s	.140 s

Table 5.5: Average latency incurred and break even times (T_B) for various increases in bandwidth.

There are many cases where an increase in bandwidth of 50% or more is possible. Escaping an interfering source, such as a microwave oven or a neighboring wireless

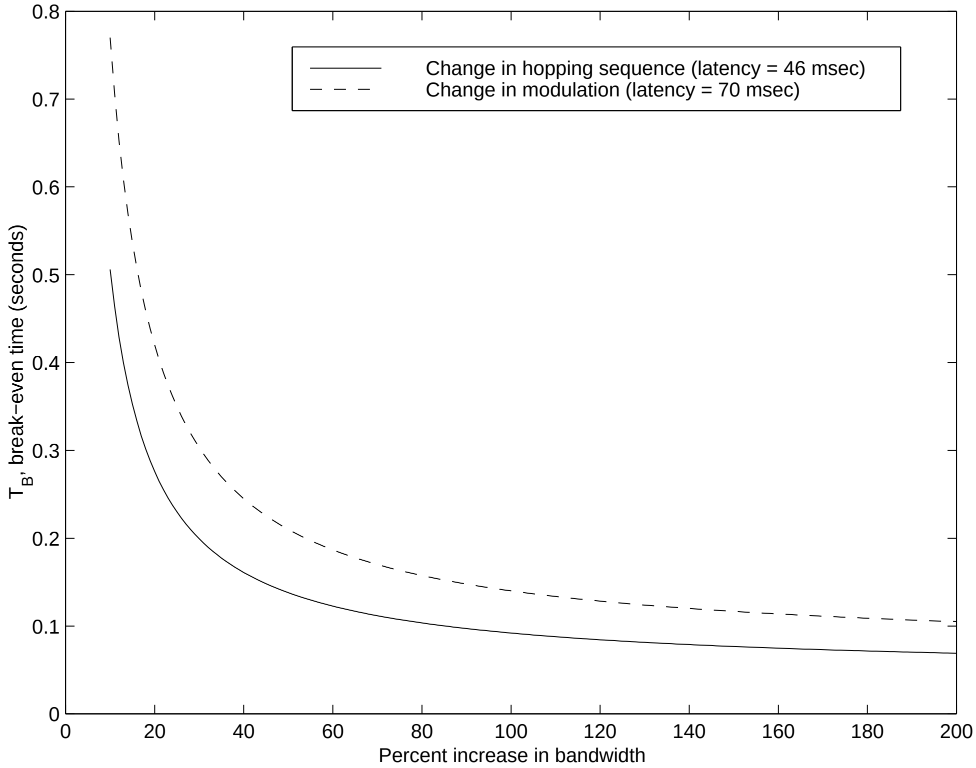

Figure 5-2: Plot of the break-even time, T_B, as a function of the percent increase in bandwidth, $\frac{B_{new}}{B_{current}} - 1$.

network, could have this impact. Also, environmental conditions can be expected in many cases to remain fairly stable for .2 seconds. Thus, it appears that the modification protocol performs well for typical situations. The data capacity that is lost during the handoff latency is quickly made up, resulting in improved performance after a very short amount of time.

Chapter 6

Conclusions and Future Work

Today's wireless networks suffer from limited performance because their wireless links are constructed with static physical layers. A network is designed once for one particular set of worst-case operating conditions. If the current conditions are better than worst-case, or if application requirements have changed, today's networks cannot modify their physical layers to improve performance. To address this problem, this thesis presents a general framework for creating adaptive physical layers by using virtual radios. By using the presented protocol to coordinate the modifications at both ends of the wireless link, a wireless network is able to adapt its physical layer while incurring at most a 70 msec pause in data transmission. Even with relatively small increases in bandwidth (eg. 10%) as a result of the modification, the performance of a network that is capable of making such a modification begins to exceed that of a static network within one second. This pause in data transmission is a small price to pay for the ability to modify the network to maintain a high level of performance.

This thesis makes three contributions toward wireless networking:

- **Application of software radio technology.** We demonstrated the ability of software radio technology to provide the necessary dynamic flexibility in the physical layer of wireless network links. Code modules corresponding to aspects of the physical layer processing, such as modulation, can be changed "on-the-fly" to provide this dynamic flexibility.

- **Protocol for the coordination of modifications.** We presented the design, analysis, and implementation of a reliable protocol for dynamically modifying network links.

- **General framework for adaptive wireless networks.** We presented the implementation of an adaptive network infrastructure that offers a general framework for modifying any aspect of the physical layer. As such, this infrastructure can be used as a basis for future research into adaptive wireless networking.

6.1 Observations and Discussion of Trends

The network infrastructure presented in this thesis relies on virtual radios to implement adaptive physical layers. Thus, the flexibility offered by such a network is based in the use of software for processing that is traditionally performed in hardware. While this reliance allows the network to take advantage of the trends in computer technology, this software approach has a disadvantage in speed. The software-based network, when operating with a data rate of 100 kbps, requires approximately 21 msec of processing time to communicate an 84 byte ping packet. A pure hardware network, such as RadioLAN, requires less than 1 msec for that same packet.

While the gap between the processing times of software and hardware-based networks is quite large today, computer technology is rapidly improving. The clock speeds of CPUs continue to double every 18 months, and new instructions set extensions, such as MMX for the Pentium and the corresponding vector floating point instructions for the Pentium III, are constantly being developed. These improvements increase the performance of virtual radios with minimal changes to the software. By riding the technology curve, the gap between the processing times of software and hardware will decrease in the future.

Adaptive networks built on virtual radio technology can also leverage the technological improvements in A/D and I/O technology. Analog-to-digital conversion technology is constantly improving, with current commercially available converters achieving sampling rates in excess of 65 MSPS. New computer I/O buses, such as AGP and the proposed double speed PCI, increase the maximum throughput between the A/D and memory. Advances in both of these areas will allow larger frequency bands to be captured for processing, enhancing the flexibility of the wireless network.

As the use wireless networks increases, the dynamic flexibility offered by adaptive physical layers will become increasingly important. Users want the same performance that they get from their wired networks with the added benefit of mobility. Performance guarantees in an unpredictable wireless environment can only be satisfied by rapidly responding to the changing conditions.

Currently, incompatible physical layer formats are being developed and used by different companies. While different physical layers will always need to exist between networks that cover vastly different areas (office coverage vs. city coverage), the inconsistencies in the physical layer among similar networks will slow the growth of wireless network usage, just as the proliferation of possible formats slowed the growth of digital cellular telephony in the United States. By rendering the interoperability problem nonexistent, adaptive networks would remove a major barrier to the widespread use of wireless networking.

6.2 Future Work

In order to effectively incorporate the adaptive physical layers presented in this thesis into intelligent wireless networks, many technologies must be developed. This section identifies three areas of future work and a fourth, RadioActive networks, that is the ultimate application of adaptive physical layer technology.

6.2.1 Effective Channel Monitoring

Accurate channel information is necessary for the effective use of adaptive physical layers. Obtaining basic information, such as available bandwidth, latency, and error rates is important, but the intelligent use of such data will have a greater impact on overall performance. For example, the Snoop protocol uses loss information from the wireless channel to augment TCP processing at wireless basestations [1]. The impact on performance of combining bit error rates and TCP acknowledgement data is considerable; it can be as great as 100% to 200%. The enormous performance gain is due to the fact that TCP interprets loss as congestion, which is often not the case for wireless channels. Adding information about bit errors allows TCP to adjust its actions accordingly.

The challenge in channel monitoring is in developing intelligent analyses of environmental data beyond snapshot observations. Trends or characteristics of the observations are important in determining the proper course of action. For example, characteristics of the errors, such as whether they are bursty or evenly distributed, are important in determining how the coding should be adapted or if packet lengths should be decreased.

6.2.2 Quantification of Performance Metrics

Recent work has addressed policy-based methods in wireless overlay networks for determining which network at a particular time is the "best" network [18]. Each reachable network is assigned a cost which is a function of the bandwidth it can offer, the power consumption of the device, and the monetary charge of using that particular network [18].

However, little work has incorporated other measures of performance, such as latency, error rates, or burstiness in the available bandwidth. These characteristics of the communication channel are important in determining which physical layer is the "best" for a particular application. For example, a videoconferencing application needs to be able to specify that a connection with low latency is more desirable than one with low error rates, with all else being equal. In order to tie the performance of the wireless links to application requirements, these other measures of performance must be quantified and included in the determination of the "best" physical layer.

6.2.3 Traffic-driven Modification Policy

Local area network traffic is inherently bursty, where a burst represents a series of packets sent across the network without the network becoming idle. Since the modification of the physical layer incurs approximately 70 msec of latency in data transmission, the optimal time to perform a modification is between bursts of traffic. Thus, we would like to wait until a burst is completed before making a modification so that some or all of the 70 msec of latency could be hidden during the network idle period.

Usually, burst lengths are assumed to follow an exponential distribution. A exponential distribution is memoryless, which means the remaining lifetime of a burst is independent of the current age of the burst [9]. If this were true, since the expected remaining length of any burst is constant, the modification decision should be made without regard toward the current traffic situation.

However, recent trace analysis has shown that the lengths of bursts on a wired local area network follow a heavy-tailed distribution [20]. A heavy-tailed distribution is one with a decreasing failure rate, which means the expected remaining lifetime increases as the current age increases [9]. Preliminary work has shown that this may also be true of wireless network traffic [7]. With heavy-tailed distributions, information about the elapsed time of the current burst provides valuable information about the expected remaining length of the burst. Long bursts are expected to continue for an even longer time, while short bursts are expected to end soon. Thus, if a modification is called for and a short burst is currently being transmitted, it may be advantageous to wait to see if the burst ends before making the modification. Conversely, if a long burst is currently being transmitted, it may be better to change now and incur the handoff latency, since it is likely that the current burst will last for a long time.

Future work in this area will involve more detailed analysis into wireless network traffic to determine the degree to which it is heavy-tailed. If the results are promising, a new decision rule for performing modifications can be developed to incorporate the current traffic analysis with current channel conditions.

6.2.4 RadioActive Networks

Adaptive physical layers provide a flexible communication infrastructure for wireless networks. Active networks provide programmability within the network to rapidly modify the nodes to adapt the higher network layers [19]. RadioActive networks draw upon the strengths of these two technologies to provide the greatest level of flexibility to optimize the entire network [4]. Information may be passed across layers in order to perform overall system optimizations. For example, quality of service could be assisted by dividing the physical layer into multiple channels with different characteristics to serve the different service qualities required. Routing algorithms could utilize information from the physical layer to route around links that are unsuitable for certain types of traffic or to conserve power by using a larger number of shorter hops.

By adding control over the physical layer to an active network that can modify the upper layers, many more performance enhancing optimizations are possible.

6.3 Conclusion

As wireless technology becomes more popular, the demand for high performance wireless networking will grow. Software radio technology, which facilitate the application of adaptive physical layers, will greatly improve the performance of wireless networks. By providing a mechanism to adapt the underlying communication channel of wireless networks, adaptive physical layers provide the first step towards bringing the ideal of reliable anytime, anywhere connectivity closer to reality.

Bibliography

[1] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. In *SIGCOMM '96*, pages 256–269, Stanford, CA, August 1996. ACM.

[2] Vanu G. Bose. *Virtual Radios*. PhD thesis, MIT, June 1999.

[3] Vanu G. Bose, Michael Ismert, Matthew Welborn, and John Guttag. Virtual Radios. *JSAC issue on Software Radios*, April 1999.

[4] Vanu G. Bose and David J. Wetherall. Radioactive networks: Freedom from worst case design. *Mobicom'99*, August 1999.

[5] Alison Brown and Barry Wolt. Digital L-Band Receiver Architecture with Direct RF Sampling. In *IEEE Postition Location and Navigation Symposium*, pages 209–216, April 1994.

[6] Howard Cheung. Frequency of a Microwave Oven, 1998. http://www.columbia.edu/ gae4/facts/HowardCheung.html.

[7] Andrew Chiu. The Distribution of Connection Durations in Wireless Links, December 1998. http://www.sds.lcs.mit.edu/ agchiu/6891.ps.

[8] Douglas E. Comer. *Internetworking With TCP/IP*, volume 1. Prentice Hall, Englewood Cliffs, NJ 07632, second edition, 1991.

[9] Mor Harchol-Balter and Allen Downey. Exploiting Process Lifetime Distributions for Dynamic Load Balancing. *ACM Transactions on Computer Systems*, 15(3):253–285, August 1997.

[10] IEEE. IEEE Std 802.11-1997. Technical report, IEEE, 1997. Working Group for Wireless Local Area Networks.

[11] Joseph Mitola III. Software Radios: Wireless Architectures for the 21st Century. http://ourworld.compuserve.com/homepages/jmitola/.

[12] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, NJ 07632, second edition, 1988.

[13] Raymond J. Lackey and Donal W. Upmal. Speakeasy: The Military Software Radio. *IEEE Communications Magazine*, 33(5):56–61, May 1995.

[14] Edward A. Lee and David G. Messerschmitt. *Digital Communication.* Kluwer Academic Publishers, second edition, 1994.

[15] Shankar Narayanaswamy, Srinivasan Sheshan, and et al. Application and Network Support for InfoPad. *IEEE Personal Communcations Magazine*, March 1996.

[16] Mark Stemm and Randy H. Katz. Vertical Handoffs in Wireless Overlay Networks. *ACM Mobile Networking, Special Issue on Mobile Networking in the Internet*, Winter 1998.

[17] Andrew S. Tanenbaum. *Computer Networks.* Prentice Hall, Englewood Cliffs, NJ 07632, second edition, 1988.

[18] Helen J. Wang, Randy H. Katz, and Jochen Giese. Policy-Enabled Handoffs Across Heterogeneous Wireless Networks. In *WMCSA 99*, New Orleans, LA, 1999.

[19] David J. Wetherall. *Service Introduction in an Active Network.* PhD thesis, MIT, February 1999.

[20] Walter Willinger, Vern Paxson, and Murad S. Taqqu. Self-Similarity and Heavy Tails: Structural Modeling of Network Traffic. In R. Adler, R. Feldman, and M.S. Taqqu, editors, *A Practical Guide to Heavy Tails: Statistical Techniques and Applications.* Birkhauser, 1998.