# PROTOCOLS AND ARCHITECTURE FOR WIRELESS AD HOC NETWORKS

VIKAS KAWADIA, Ph.D.
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, 2004
Professorssor P.R. Kumar, Adviser

The subject of this dissertation is ad hoc wireless networks. In such networks packets are relayed over multiple hops to reach their destination. In order to operate ad hoc networks several protocols, for media access control, power control, routing, and transport are needed. This dissertation is concerned with the development and evaluation of such protocols, and the associated architecture of the protocol stack. We take a holistic approach to the problems in ad hoc networks.

Transmission power control is important because of the fundamental nature of the wireless network that it is interference limited. Transmission power control has the potential to increase a network's traffic carrying capacity, reduce energy consumption, and reduce the end-to-end delay. We start by postulating general design principles for power control based on the effect of transmit power on various performance metrics. These are used to design a set of protocols which attempt to optimize different performance metrics, as all the metrics cannot be simultaneously optimized in general. Some of these protocols have been implemented in the Linux kernel in an architecturally appropriate manner. Extensive testing was not possible due to the limitations of the current generation of hardware, and so performance results obtained through NS2 simulations are used to illustrate the potential benefits of the power control protocols.

Next we investigate the transport layer in static multihop wireless networks. Based on an experimental study and statistical analysis, we make recommendations on modifications to TCP for use in ad hoc wireless networks. We suggest clamping the congestion window,

turning off RTS-CTS and turning on the TCP selective acknowledgment on, among other things. Our recommendations are based on experimental testing of all possible combinations of these and other modifications on nine different topologies of up to six hops, with several runs of each experiment. The results are analyzed through statistical techniques to arrive at the recommendations suggested above. This work has also resulted in the development of a set of tools to automate the process of experimentation, data collection, processing and presentation.

Our work at the routing layer is motivated by the difficulty in implementing on-demand routing protocols for ad hoc networks in common operating systems. It provides an API and its implementation in Linux as a user-space library (ASL), which facilitates implementation of on-demand routing protocols. It has been used to implement a few routing protocols.

Then this dissertation addresses the important issues of cross-layer design and system architecture. We make the case for a cautionary approach to designing cross-layer schemes and protocols. Some examples are given to illustrate how cross-layer design can lead to unintended interactions which could actually degrade system performance.

Given the increased interest in cross-layer design, we provide the outline of a general approach to protocols and layering. A central feature of our suggested model is that each layer have its own sets of addresses, and the interaction between the layers be enforced strictly through inter-layer services. We illustrate how the generalized stack solves many problems cleanly.

The dissertation concludes by describing the design and implementation of an ad hoc networking software module, which provides certain basic services necessary for configuring and operating ad hoc networks.

# PROTOCOLS AND ARCHITECTURE FOR WIRELESS AD HOC NETWORKS

BY

VIKAS KAWADIA

B.Tech, Indian Institute of Technology, Bombay, 1999
M.S., University of Illinois at Urbana-Champaign, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

# PROTOCOLS AND ARCHITECTURE FOR WIRELESS AD HOC NETWORKS

**Approved by**
**Professor P. R. Kumar (Chairman)**
**Professor Lui Sha**
**Professor R. S. Sreenivas**
**Professor Nitin H. Vaidya**
**Professor Venugopal V. Veeravalli**

_____

_____

_____

_____

_____

# ABSTRACT

The subject of this dissertation is ad hoc wireless networks. In such networks packets are relayed over multiple hops to reach their destination. In order to operate ad hoc networks several protocols, for media access control, power control, routing, and transport are needed. This dissertation is concerned with the development and evaluation of such protocols, and the associated architecture of the protocol stack. We take a holistic approach to the problems in ad hoc networks.

Transmission power control is important because of the fundamental nature of the wireless network that it is interference limited. Transmission power control has the potential to increase a network's traffic carrying capacity, reduce energy consumption, and reduce the end-to-end delay. We start by postulating general design principles for power control based on the effect of transmit power on various performance metrics. These are used to design a set of protocols which attempt to optimize different performance metrics, as all the metrics cannot be simultaneously optimized in general. Some of these protocols have been implemented in the Linux kernel in an architecturally appropriate manner. Extensive testing was not possible due to the limitations of the current generation of hardware, and so performance results obtained through NS2 simulations are used to illustrate the potential benefits of the power control protocols.

Next we investigate the transport layer in static multihop wireless networks. Based on an experimental study and statistical analysis, we make recommendations on modifications to TCP for use in ad hoc wireless networks. We suggest clamping the congestion window, turning off RTS-CTS and turning on the TCP selective acknowledgment on, among other things. Our recommendations are based on experimental testing of all possible combinations of these and other modifications on nine different topologies of up to six hops, with several runs of each experiment. The results are analyzed through statistical techniques to arrive at the

recommendations suggested above. This work has also resulted in the development of a set of tools to automate the process of experimentation, data collection, processing and presentation.

Our work at the routing layer is motivated by the difficulty in implementing on-demand routing protocols for ad hoc networks in common operating systems. It provides an API and its implementation in Linux as a user-space library (ASL), which facilitates implementation of on-demand routing protocols. It has been used to implement a few routing protocols.

Then this dissertation addresses the important issues of cross-layer design and system architecture. We make the case for a cautionary approach to designing cross-layer schemes and protocols. Some examples are given to illustrate how cross-layer design can lead to unintended interactions which could actually degrade system performance.

Given the increased interest in cross-layer design, we provide the outline of a general approach to protocols and layering. A central feature of our suggested model is that each layer have its own sets of addresses, and the interaction between the layers be enforced strictly through inter-layer services. We illustrate how the generalized stack solves many problems cleanly.

The dissertation concludes by describing the design and implementation of an ad hoc networking software module, which provides certain basic services necessary for configuring and operating ad hoc networks.

The Road goes ever on and on
Down from the door where it began.
Now far ahead the Road has gone,
And I must follow, if I can,
Pursuing it with eager feet,
Until it joins some larger way
Where many paths and errands meet.
And whither then? I cannot say. [1]

[1] Bilbo Baggins before embarking on his adventure in *The Lord of the Rings,* by J. R. R. Tolkien.

# ACKNOWLEDGMENTS

I earnestly thank my adviser Professor P. R. Kumar for his excellent advice and guidance. The immense trust he placed in my abilities was always a great source of motivation.

I thank all committee members for their comments and advice during the preliminary examinations. Special thanks to Professor Nitin Vaidya for the discussion and advice on several topics. Several other people have been helpful directly or indirectly. They include Arvind Giridhar, Rajat Prakash, Vivek Raghunathan, Supratim Deb, Charuhas Pandit, Girish Baliga, Sujay Sanhgavi, Swetha Narayanswamy, Binita Gupta, Paromita Mukherjee, Feng Xue, Robert Rozovosky, Nitin Gupta, Ashish Agrawal, Scott Graham, Rong Zheng, Apu Kapadia, and others I might have missed.

On a personal note, CSL, especially room 139, has been a fabulous working environment. I also thank Francie Bridges and Becky Lonberger for making the stay in CSL ultraconvenient.

Outside of work, my stay in Champaign has been truly memorable. I got introduced to the wonderful game of squash which has been a real life support system. Some of the adventure trips with the gang here are unforgettable. In addition to the people already mentioned, Zehra Sura, Lakhsmi Subramanian, Saurabh Tavildar, Anurag Ganguly, Siddarth Mallik, Yingquan Wu, Mike Chen, Srinivas Shakkottai, and others have provided excellent company. Dilip and Aradhana Chajjed, and Mrs. Jaya Kumar deserve special thanks for the great food and support. They have been very friendly local guardians.

The more important thanks are reserved for the last. Many thanks to my parents and sister for their constant support, concern, and motivation. Finally, thanks to my love Preeti for everything, and everything that is beyond words.

# TABLE OF CONTENTS

# LIST OF FIGURES

**Figure** **Page**

# LIST OF TABLES

**Table** **Page**

# ACRONYMS AND ABBREVIATIONS

**ACK** – Acknowledgment

**AODV** – Ad hoc on-demand distance vector

**ANOVA** – Analysis of variance

**API** – Application Programming Interface

**ARP** – Address resolution protocol

**BGP** – Border gateway protocol

**CDMA** – Code division multiple access

**CGI** – Common gateway interface

**CMOS** – Complementary metal-oxide-semiconductor (A type of field effect transistor)

**CSMA/CA** – Carrier sense multiple access with collision avoidance

**CTS** – Clear to send

**DHCP** – Dynamic host configuration protocol

**DSDV** – Destination sequenced distance vector

**DSR** – Dynamic source routing

**ESSID** – Extended service set identifier

**FCC** – Federal communications commission

**FDMA** – Frequency division multiple access

**GPS** – Global positioning system

**ICMP** – Internet control message protocol

**IP** – Internet Protocol

**IETF** – Internet engineering task force

**LAN** – Local area network

**MAC** – Media access control

**MACA** – Multiple access with collision avoidance

**MANET** – Mobile ad hoc network

**MSS** – Maximum segment size

**MTU** – Maximum transmission unit

**NAV** – Network allocation vector

**OLSR** – Optimized link state routing

**OS** – Operating system

**OSPF** – Open shortest path forwarding

**OSI** – Open systems interconnection

**PLCP** – Physical layer convergence procedure

**QoS** – Quality of service

**RED** – Random early detection

**RF** – Radio frequency

**RFC** – Request for Comments

**RIP** – Routing information protocol

**RTO** – Retransmission timeout

**RTS** – Request to send

**RTT** – Round trip time

**SACK** – Selective acknowledgments

**SNR** – Signal to noise ratio

**TCP** – Transmission control protocol

**TDMA** – Time division multiple access

**TORA** – Temporally ordered routing algorithm

**UDP** – User datagram protocol

**WEP** – Wired equivalent privacy

**ZRP** – Zone routing protocol

# CHAPTER 1

# INTRODUCTION

A wireless ad hoc network is a decentralized network of nodes with radios, possibly mobile, sharing a wireless channel and asynchronously sending packets to each other, generally over multiple hops. The most notable characteristics of an ad hoc network are a lack of infrastructure, multihop communication by cooperative forwarding of packets, distributed coordination among nodes, dynamic topology, and the use of a shared wireless channel.

The potential for deployment of ad hoc networks exists in many scenarios, for example, in situations where infrastructure is infeasible or undesirable, like disaster relief, sensor networks, etc. Ad hoc networks also have the potential of realizing a free, ubiquitous, omnipresent communication network for the community. We provide a layer-by-layer summary of the progress in ad hoc networks.

Progress at the physical layer has been facilitated by advances in both digital CMOS technology as well as analog RF circuits, which have made powerful RF transceivers inexpensive. Advances in digital communications have also played an important role by providing spread spectrum technologies, better modulation schemes, improved coding schemes, etc. As a result, wireless cards supporting data rates in excess of 50 Mb/s are now affordable.

Medium access in ad hoc networks is a complex problem. Multiple access schemes popular in cellular networks are not easy to implement in ad hoc networks because of the need to dynamically allocate resources efficiently to allow spatial reuse. Dynamic assignment of frequency bands for FDMA, time slots for TDMA, or spreading codes for CDMA, are difficult because of node mobility and the consequent need to keep track of resources in a distributed fashion. Consequently, random access schemes where all nodes have anytime, anywhere access to the

channel, have emerged as the preferred choice. In this model, simultaneous reception of two or more strong signals at the receiver is considered a collision. CSMA/CA is the principal medium access method adopted for a shared wireless channel. However, CSMA/CA does not solve problems associated with *hidden terminals* and *exposed terminals*, which have been tackled in MACA [1], and several extensions (see e.g., [2]), through a series of handshakes. The current industry standard for the physical layer and the link layer in wireless LANs and ad hoc networks, IEEE 802.11 [3], uses a combination of CSMA/CA and handshaking to solve the media access problem. IEEE 802.11 has been immensely popular for wireless LANs but its operation in ad hoc mode presents many problems. Still, it is currently the only choice if one is constrained to use commercial off-the-shelf equipment for ad hoc networking.

The wireless medium is an interference limited medium. Reducing the interference caused by various transmissions is critical for the efficiency and scalability of any wireless system. This motivates transmission power control, which is a very complex problem for ad hoc networks. In cellular networks power control is a physical layer problem, where each transmitter adjusts its power level to achieve equal SNR at a common receiver that is the base station. A feasible choice of power levels exists for all the transmitters, and iterative algorithms exist to achieve the solution [4, 5]. Ad hoc networks do not admit such a solution for power control, since there are many receivers, and in general it is not possible to simultaneously equalize SNR at several receivers. One goal of power control in ad hoc networks is to increase network capacity by increasing spatial reuse. Other possible goals are to save battery power and reduce end-to-end delay. Numerous approaches, for example [6, 7], attempt to perform power control at the MAC layer by adjusting the power level of every packet to guarantee an acceptable SNR at a given receiver, using feedback from the receiver. Another class of approaches, called topology control, vary transmit power at the network layer to favorably alter topology (since power decides range) according to certain metrics [8, 9]. There are other routing and media access schemes which focus on energy savings only, by using a variety of performance measures [10]. This spectrum of schemes traversing all the layers illustrates the complexity of choices regarding power control in ad hoc networks. Which layer does power control belong to? How does it impact the other layers? What is a suitable architecture for power control? These are questions that need to be answered if power control protocols are to be widely adopted.

The network layer provides the critical service of routing packets between remote nodes. The dynamic nature of ad hoc networks coupled with the lack of hierarchy makes routing very difficult in mobile ad hoc networks. However, many of the popular routing schemes, e.g., those which use the number of hops to the destination as the metric, do not work well even in static ad hoc networks. This is because the dynamic and interference limited nature of the wireless channel implies that the quality of links selected on a route is also important, and not just the number of links. Thus, the fundamental issue of selecting appropriate metrics for routing in wireless ad hoc networks is not yet well understood.

Even the popular routing protocols have been studied mostly via simulations. Few implementations exist for real testbeds. The primary reason for this is the lack of support in current operating systems for implementing various features required by these protocols. Such operating system support is important if routing protocols are to be widely deployed.

At the transport layer, the dominant protocol in wired networks is TCP, and the issue is whether it is appropriate for wireless ad hoc networks. TCP has been studied extensively for last-hop wireless networks, and many effective solutions have been suggested [11–13]. They essentially provide a mechanism to distinguish between random losses and congestion losses through explicit or implicit support from the link layer, since TCP cannot make this distinction. However, most media access schemes in ad hoc networks provide reliable delivery using hop-by-hop acknowledgments, thus shielding TCP from random losses at the link layer. Hence, problems with TCP in ad hoc networks are different from those in last-hop wireless networks.

One important issue is TCP's reaction to route failures or instabilities caused by mobility. Link layer losses may also occur in-spite of link layer ACKs, and do affect TCP dynamics. Some solutions for ad hoc networks include Ad Hoc TCP (ATCP) [14], Ad Hoc TCP (ADTCP) [15], TCP-Feedback (TCP-F) [16] and TCP with Detection of Out-of-Order and Response (TCP-DOOR) [17]. They differ in the mechanism used to differentiate between different type of losses and dealing with them. But there might be problems with the functioning of TCP even in static ad hoc networks. These issues need to be throughly investigated.

We have summarized above some issues related to the individual layers of the network stack. The architecture of the stack as a whole is also important. The popular TCP/IP stack for wired networks is often taken as the starting point for designing ad hoc networks. However, violating

the layering often provides a performance improvement in wireless networks. This has led to the blooming field of cross-layer design. Keeping in mind the entirety of the network stack, these cross-layer proposals need to be carefully examined for their effects on various parts of the stack.

## 1.1 Outline

In this work, we identify and address several problems in protocol design for ad hoc networks. This dissertation takes a pragmatic yet principled approach to designing and building ad hoc networks. The problems are approached in a holistic manner. The protocols we develop are implemented and tested in real systems to the extent possible. This ensures that due importance is given to both system and software architecture. Contributions are made in the form of design principles, protocols, implementations in real systems, simulational and experimental studies, architectural suggestions for the stack as a whole, and several tools and libraries.

Chapter 2 deals primarily with the power control problem. We conceptualize the power control problem and exhibit a design situated at the network layer. We advocate certain design principles for power control in ad hoc networks. Then the CLUSTERPOW protocol is presented which provides a joint solution for power control, routing and clustering at the network layer. It generalizes the earlier COMPOW protocol which provides routes based on a common power level throughout the network. CLUSTERPOW provides increased spatial reuse, and hence increases network capacity. Then we present another protocol, called MINPOW, which provides an optimal solution for energy consumption, by using energy consumption as the metric in the distributed Bellman Ford algorithm. The protocols are proved to be loop free, and other properties of interest are pointed out. The next protocol called LOADPOW constitutes a cross-layer scheme to reduce the end-to-end delay. We provide an implementation of COMPOW, CLUSTERPOW and MINPOW in Linux. The software architecture provided is general enough to be useful for implementing other power control schemes as well. The protocols have also been implemented in Network Simulator 2 (NS2) [18], and simulation results indicate the potential benefits of power control.

Chapter 3 is a thorough experimental investigation of TCP and some modifications to TCP for ad hoc networks. Several experiments are conducted, and the results are statistically analyzed to check their significance. Based on the experimental investigations and statistical testing we present recommendations in the form of TCP modifications, and setting of MAC parameters for ad hoc networks. In addition, several other interesting observations are made concerning the IEEE 802.11 MAC protocol and its interaction with TCP. Some of the techniques used in the experiments, (e.g., using copper tape to reduce the range of the wireless cards) may be useful in general. Also, the tools developed for setting up the experiments, collecting the data, processing the data and making the data available in a useful form should be of independent interest in their own right.

Chapter 4 studies the system design issues associated with implementation of ad hoc routing protocols. Existing operating system primitives are shown to be insufficient for implementing on-demand routing protocols. We propose an Application Programming Interface (API) for operating systems to enable ad hoc routing, and provide an implementation for the API in Linux, as a user-space library called Ad Hoc Support Library (ASL). ASL makes efficient implementation of ad hoc routing protocols possible. For instance, the AODV protocol has been implemented in [19] using ASL.

Chapter 5 addresses the popular field of cross-layer design, and presents a cautionary perspective. We illustrate the benefits of a good architectural design, and possible negative effects of cross-layer design through examples.

Chapter 6 examines the problem of layering and protocols, with the current TCP/IP stack as an example. We provide a generalized notion of layering and protocols, which provides clean architectural solutions to many problems in wireless networks. The dissertation concludes in Chapter 7.

# CHAPTER 2

# THE TRANSMIT POWER CONTROL PROBLEM

The wireless medium is a shared medium. Every transmission causes interference in the surrounding area. Successful reception of packets is possible only if this interference is within some limits. Thus, interference is a key feature of the wireless medium and fundamentally affects the traffic carrying capability of the wireless network. One of the effective mechanisms of controlling this interference is by controlling the transmission power. This motivates the transmit power control problem, which is the topic of this chapter.

The transmit power control problem in wireless ad hoc networks is that of choosing the transmit power for each packet in a distributed fashion at each node. We begin by making the case that power control is a challenging example of a design problem that cuts across several layers. The problem is complex since the choice of the power level fundamentally affects many aspects of the operation of the network:

i. The transmit power level determines the quality of the signal received at the receiver.

ii. It determines the range of a transmission.

iii. It determines the magnitude of the interference it creates for the other receivers.

Because of these factors, power control affects several layers in the OSI hierarchy:

iv. Power control affects the physical layer (due to i).

v. It affects the network layer since the transmission range affects routing (due to ii).

vi. It affects the transport layer because interference causes congestion (due to iii).

Since each layer has an impact on some performance metric, power control has a multi-dimensional effect on the performance of the whole system. This impact is manifested in several ways:

i. The power levels determine the performance of medium access control since the contention for the medium depends on the number of other nodes within range.

ii. The choices of power levels affect the connectivity of the network [20, 21], and consequently the ability to deliver a packet to its destination.

iii. The power level affects the throughput capacity of the network [22].

iv. Power control affects the contention for the medium, as well as the number of hops, and thus the end-to-end delay.

v. Transmit power also affects the important metric of energy consumption, which directly affects the battery life, a very important consideration for mobile, portable devices.

Any design of a power control protocol must also be vigilant to the effect it can have on other well-established protocols and the functioning of the associated layers. The reason is that the assumption of fixed power levels is so ingrained into the design of many protocols in the OSI stack that changing the power levels can easily result in their malfunctioning. We illustrate this with a few examples.

i. Changing power levels can create unidirectional links, which can happen when a node $i$'s power level is high enough for a node $j$ to hear it, but not vice versa.

ii. Bidirectionality of links is implicitly assumed in many routing protocols. For example, distributed Bellman-Ford, the basis of many minimum hop routing protocols, uses the dynamic programming recursion: $V_{ij} = \min_k [c_{ik} + V_{kj}]$, where $c_{ik}$=1 if $k$ is is a neighbor of $i$ and $\infty$ otherwise, and $V_{ij}$ is the minimum number of hops from $i$ to $j$. If $i$ can hear $k$, then node $i$ may hear the distance $V_{kj}$ advertised by node $k$, but $1+V_{kj}$ is not the distance from $i$ to $j$ via $k$, if $k$ cannot hear $i$. The problem is that the notion of a "neighbor" ceases to be a symmetric notion and $c_{ik} \neq c_{ki}$.

iii. Medium access protocols such as IEEE 802.11 [3] implicitly rely on bidirectionality assumptions. For example, a CTS from $j$ silences only those nodes which can hear $j$, but there may be other higher powered nodes that $j$ can hear. ACKs also assume bidirectional links.

iv. Various protocols employ route reversals, e.g., route-reply packets in AODV [23] and DSR [24] reverse the route followed by the route request packets.

Transmit power control is therefore a prototypical cross-layer design problem affecting all layers of the protocol stack from physical to transport, and affecting several key performance measures, including the trinity of throughput, delay and energy consumption. Cross-layer design, in general, should be approached holistically with some caution, keeping in mind longer term architectural issues [25].

The first issue that arises is where in the network architecture should power control be located. Its resolution requires an appreciation of the issues involved at each layer.

The second issue that arises is how exactly to choose the power level. This solution needs to be guided by its impact on multiple performance measures, which in turn requires a theoretical understanding of the impact of power control.

The final issue that arises is the issue of the software architecture for the implementation. We need to take into account the software organization of the IP stack, and the interplay between the kernel, user-space applications, and the firmware on the wireless cards. The solution also needs to be appropriately modularized to allow future changes in routing protocols without redesigning the entire power control solution.

Given this complex web of interactions, we begin by distilling a few basic design principles to guide our design process for power control [26]. Then we propose some protocols which attempt to achieve several design objectives and perform several optimizations simultaneously. The COMPOW protocol [27] attempts to increase network capacity, while meeting the needs of several other layers by choosing a common power level for use by all the nodes throughout the network. The CLUSTERPOW protocol [28] relaxes this constraint and provides a joint solution to the power control, clustering and routing problem, again with the goal of maximizing network capacity. The tunneled CLUSTERPOW protocol develops a more sophisticated way

of achieving a finer optimization for network capacity, at the cost of greater implementation complexity. The MINPOW protocol achieves a globally optimal energy consumption solution for awake nodes, but may or may not increase network capacity depending on the wireless hardware. The LOADPOW protocol attempts to reduce end-to-end delay by using higher power levels, when the network load is low.

We also present software architectural designs for cleanly implementing these protocols. We have implemented COMPOW, CLUSTERPOW, and MINPOW. Tunneled CLUSTERPOW requires considerably more implementation effort and was not implemented, while LOADPOW could not be implemented as it needs changes in the MAC protocol, which resides in the firmware of the wireless card, which is not accessible.

Experimental performance evaluations were anyway not possible for any of the protocols due to hardware limitations, which is essentially designed for changing power levels at startup. Thus, for quantitative comparisons we have also implemented some of these protocols in the NS2 simulator, which interestingly turned out to require more effort than the real implementations in the kernel.

The rest of the chapter is organized as follows. Section 2.1 describes and justifies a few design principles for power control. The next five sections describe the different power control protocols, prove various properties and describe the implementation of the protocols. Section 2.8 presents the simulation results in NS2 and gives some details on the implementation of the protocols in NS2. The chapter concludes with a brief literature survey in Section 2.9.

## 2.1 Design Principles For Power Control

We begin our exploration by presenting some design principles for power control.

I. **To increase network capacity it is optimal to reduce the transmit power level.** Any transmission causes interference in the surrounding region due to the shared nature of the wireless channel. The area of this interference is reduced by reducing the transmission range, or the power level. Low power levels, however, result in a larger number of shorter hops, thus increasing the relaying burden on a node. For a transmission range of $r$,

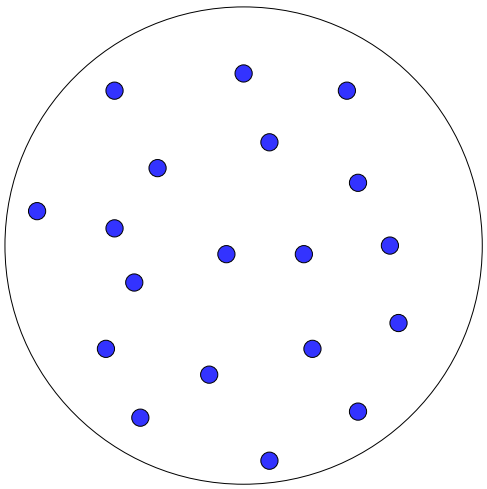


**Figure 2.1** A disk of area A $m^2$ containing $n$ nodes.

the area of the interference is proportional to $r^2$, whereas the relaying burden (i.e., the number of hops) is inversely proportional to $r$. The area consumed by a packet is thus proportional to $r$ ($r^2$ for 3-D networks), implying that reducing the transmit power level increases network capacity. This argument holds fairly generally, and not just for the asymptotic case when the number of nodes tends to infinity. However, the argument does not hold for one-dimensional networks, where the capacity gains by interference reduction are exactly offset by the increased relaying burden.

We now present the above argument in a more quantitative form, simplified from [22] and also presented in [27]. Consider a domain of area $A$ $m^2$, taken as a disk for simplicity of discussion, and containing $n$ nodes as shown in Figure 2.1. Suppose that each node can transmit at $W$ bits/s, and that the range of each node is $r$ meters. To model interference, let us simply suppose that for a node $R$ to successfully receive a packet from node $T$, it has to lie within a distance $r$ from $R$, and there can be no other simultaneous transmitter within a distance $(1 + \Delta)r$ of $R$. This model, called the protocol model, is depicted in Figure 2.2. The quantity $\Delta > 0$ captures the notion of allowing only weak interference. The protocol model is reasonably accurate for current wireless technologies like IEEE 802.11b, which employ carrier sensing. Suppose each source node has a destination node to which it wishes to send data at a rate of $\lambda$ bits/s. Suppose also that the average

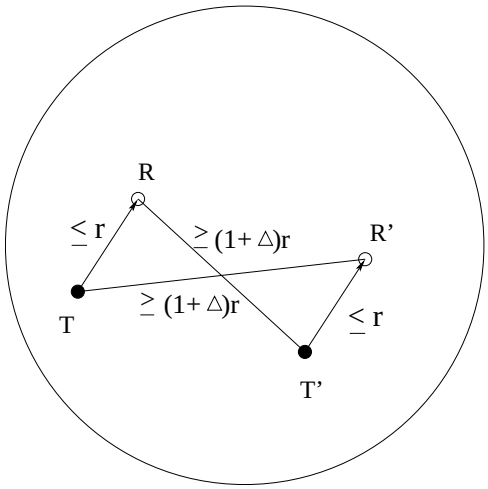



**Figure 2.2** The protocol model of interference.

distance between a source and a destination is $L$ meters. The question we will investigate is: How does $\lambda$ vary with $r$?

For simplicity, assume a slotted operation (otherwise the capacity is even less). The average number of hops per source-destination pair is at least $\frac{L}{r}$. Thus the $n$ nodes, in totality, require $\frac{Ln}{r}$ hops. Each hop requires $\lambda$ bits/s. Hence, a total of at least $\frac{Ln\lambda}{r}$ bits/s *needs to be transmitted by all the transmitters on average* in order to carry the per node throughput of $\lambda$ bits/s.

Now let us examine how much can actually be transmitted. Consider two simultaneous transmissions, one from $T$ to $R$, and another from $T'$ to $R'$, as shown in Figure 2.2. For $R'$ to hear $T'$, we need $|T' - R'| \leq r$ (where $|T' - R'|$ denotes the distance between $T'$ and $R'$). On the other hand, to avoid interference we need $|T' - R| \geq (1 + \Delta)r$. From the triangle inequality, we see that $|T' - R'| + |R' - R| \geq |T' - R| \geq (1 + \Delta)r$. Hence $|R' - R| \geq (1 + \Delta)r - |T' - R'| \quad \geq \quad (1 + \Delta)r - r = \Delta r$. Thus, disks of radius $\frac{\Delta r}{2}$ around $R$ and $R'$ are disjoint, as shown in Figure 2.3.

The interpretation is that each transmission "consumes" a "wireless footprint" of area $\frac{\pi\Delta^2 r^2}{4}$. Thus, we observe another important fact: *Area is a valuable resource too in ad hoc networks*, in addition to the shared radio spectrum. Note that the total area of the domain is $A$ m$^2$. At least a fourth of the consumed area must lie in the domain even if the receiver is on the boundary of the domain. Thus, at most $A/\frac{\pi\Delta^2 r^2}{16} = \frac{16A}{\pi\Delta^2 r^2}$ transmissions

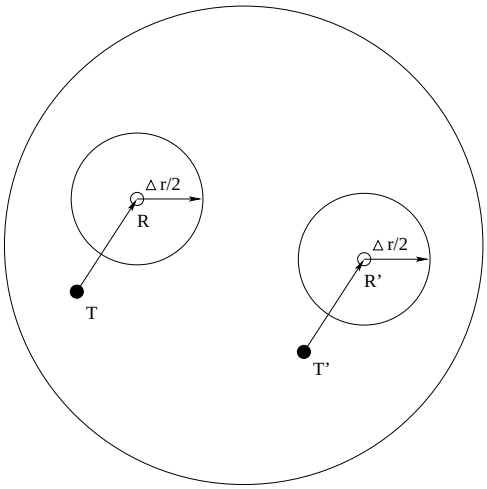



**Figure 2.3** Wireless transmissions consume area.

are simultaneously feasible. Each transmission can be at $W$ bits/s. Hence the total number of bits/s that *can be transmitted is no more than* $\frac{16AW}{\pi\Delta^2 r^2}$.

Thus, to sustain a per node throughput of $\lambda$ bits/s, we need $\frac{Ln\lambda}{r} \leq \frac{16AW}{\pi\Delta^2 r^2}$, i.e.,

$$\lambda \leq \frac{16AW}{\pi\Delta^2 L} \cdot \frac{1}{nr} \text{bits/s}.$$

Due to the reciprocal dependence of the right hand side on $r$, decreasing $r$ increases the network capacity.

II. **Reducing the transmit power level reduces the average contention at the MAC layer.**

Changing the range changes the number of one-hop neighbors that each node has and thus, the number of neighbors it has to contend with for media access. At the same time, changing the range changes the number of hops in a route and thus, the relaying burden that each node has to carry and consequently the amount of traffic that each node has to transmit. The following argument shows that the net radio traffic in contention range is proportional to $r$, which is minimized by reducing $r$ [27].

Suppose each node has traffic of rate $\lambda$ bits/s that it wants to to send to a destination at an average distance of $L$ meters away. Suppose also that each node can transmit at $W$ bits/s, and that there are a total of $n$ nodes randomly placed in a disk of area $A$ m$^2$. Note that the number of hops per route from a source node to its destination node is $\frac{L}{r}$

on average. Hence this is the number of relay nodes each origin-destination pair requires. Thus each node needs to transmit $\frac{L\lambda}{r}$ bits/s on average to support the relaying burden. Now note that a node has on average $\frac{\pi r^2 n}{A}$ neighbors within a distance of $r$ from it. These nodes are essentially collocated and can be regarded as sharing the common channel of $W$ bits/s. These $\frac{\pi r^2 n}{A}$ nodes need to transmit, on average, at a total rate of $\frac{\pi n L \lambda r}{A}$ bits/s, over a common channel which can carry $W$ bits/s. Since the channel of $W$ bits/s is fixed, the contention is reduced when the total bit rate of $\frac{\pi n L \lambda r}{A}$ bits/s is reduced. This clearly happens when $r$ is reduced, i.e., at a low value of range. Thus, the MAC contention is also minimized when we choose a low common power level.

III. **The impact of power control on total energy consumption depends on the energy consumption pattern of the hardware.**

Power consumption for communication has three components: $P_{Rx_{elec}}$, the power consumed in the receiver electronics for processing; $P_{Tx_{elec}}$, the power consumed by the transmitter electronics for processing; and $P_{Tx_{Rad}}(p)$, the power consumed by the power amplifier to transmit a packet at the power level $p$, where $p$ is the actual power that is radiated in the medium. Also, define $P_{Idle}$ to be the power consumed when the radio is on but no signal is being received, and finally, let $P_{Sleep}$ be the power consumed when the radio is turned off. Based on the relative values of these parameters, we can distill the following three principles regarding energy consumption:

(a) *If the energy consumed for transmission $P_{Tx_{Rad}}(p)$ dominates, then using low power levels is broadly commensurate with energy-efficient routing for commonly used inverse $\alpha^{th}$ law path loss models, with $\alpha \geq 2$.*

A common model assumes that path loss in the medium follows an inverse $\alpha^{th}$ law with $\alpha \geq 2$, i.e., the received power at a distance $\rho$ from a transmitter using a power level $p$ is $\frac{cp}{\rho^\alpha}$, where $c$ is a constant. Suppose that in order to receive a packet the received power level must be at least $\gamma$, i.e., $\frac{cp}{\rho^\alpha} \geq \gamma$. Then the needed transmitter power level is at least $\frac{\gamma \rho^\alpha}{c}$. Thus, if a route from a source to a destination consists of $h$ hops, of distances $\rho_i$, $i = 1, 2, \ldots, h$, then the power cost of the route is $\frac{\gamma}{c} \sum_{i=1}^{h} \rho_i^\alpha$. We can ignore the scaling $\frac{\gamma}{c}$ and just fix the power cost of the route to be $\sum_{i=1}^{h} \rho_i^\alpha$.

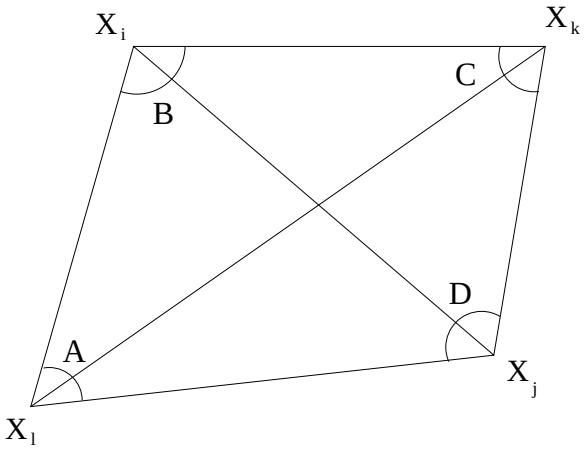



**Figure 2.4** Suppose two edges cross.

Now consider a planar domain within which are $n$ nodes at locations $X_1$, $X_2, \ldots, X_n$. For a given source node $X_s$ and destination node $X_d$, let path $p = (X_0 = X_s, X_1, X_2, \ldots, X_{h-1}, X_h = X_d)$ be a power optimal path if it minimizes $\sum_{i=1}^{h}(X_i - X_{i-1})^\alpha$ over all $X_1, \ldots, X_{h-1}$ and all $h$.

Consider now a graph $G$ formed only from edges which lie along some power optimal path from some source node $X_s$, $1 \leq s \leq n$, to some destination node $X_d$, $1 \leq d \leq n$ and such that $G$ provides a power optimal route between any two nodes.

**Lemma:** For every $\alpha \geq 2$, the graph $G$ can be chosen as a planar graph with straight line edges, i.e., no two edges cross each other. The graph for any $\alpha > 2$ can be chosen as a subgraph of that for $\alpha = 2$.

**Proof:** First consider the case $\alpha = 2$. Suppose to the contrary that there are edges $(X_i, X_j)$ and $(X_k, X_\ell)$ which cross, as in Figure 2.4. Then since $(X_i, X_j)$ is power optimal, $(X_i - X_\ell)^2 + (X_\ell - X_j)^2 \geq (X_i - X_j)^2$. If equality holds above, then we can delete the edge $(X_i, X_j)$ and replace it with the route $\{(X_i, X_\ell), (X_\ell, X_j)\}$. So without loss of generality, assume that strict inequality holds above. Then, from Euclidean geometry, $\angle A < 90°$. Similarly, $\angle B < 90°$, $\angle C < 90°$, and $\angle D < 90°$, which is of course impossible in a quadrilateral. Hence, no two edges in $G$ can cross, which implies that $G$ is planar.

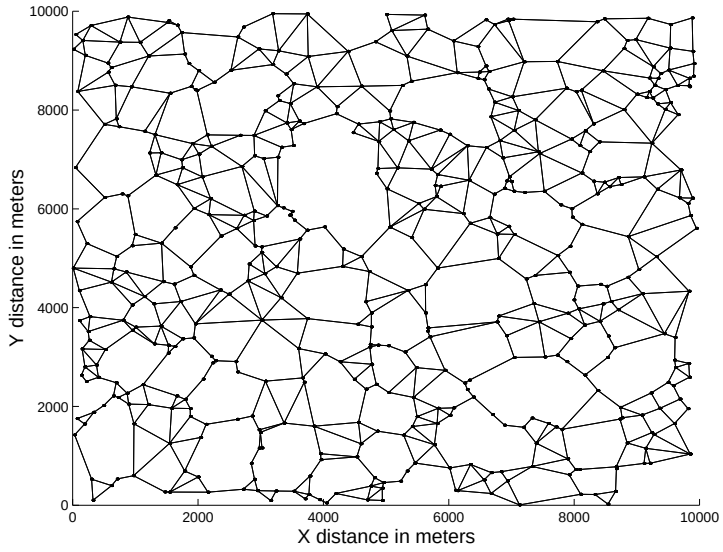



**Figure 2.5** The graph of links lying along power-optimal routes.

Now turn to the case $\alpha > 2$. We compare this with the case $\alpha = 2$, and claim that when $\alpha > 2$ no new edges need to be added (while some edges may possibly be removed). Suppose some edge $(X_p, X_q)$ has necessarily to be added. Then if $(X_p = X_{m_0}, X_{m_1}, \ldots, X_{m_{n-1}}, X_{m_n} = X_q)$ was the earlier power optimal path for $\alpha = 2$,

$$
\begin{aligned}
|X_p - X_q|^\alpha &= (|X_p - X_q|^2)^{\frac{\alpha}{2}} && (2.1)\\
&\geq (\sum_{r=1}^{n} |X_{m_r} - X_{m_{r-1}}|^2)^{\alpha/2}) && (2.2)\\
&\geq \sum_{r=1}^{n} |X_{m_r} - X_{m_{r-1}}|^\alpha && (2.3)
\end{aligned}
$$

which contradicts the need to add the edge $(X_p, X_q)$. Inequality (2.2) follows because $(X_p = X_{m_0}, X_{m_1}, \ldots, X_{m_{n-1}}, X_{m_n} = X_q)$ is optimal for $\alpha = 2$. The reason for inequality (2.3) is that whenever $y_r \geq 0$, and $\sum_r y_r =: Y$, then $\frac{y_r}{Y} \leq 1$, and so $\sum_r (\frac{y_r}{Y})^{\frac{\alpha}{2}} \leq \sum_r \frac{y_r}{Y} = 1$. So $\sum_r y_r^{\frac{\alpha}{2}} \leq Y^{\frac{\alpha}{2}} = (\sum_r y_r)^{\frac{\alpha}{2}}$. □

The following simulation, motivated by one in Shepard [29], was obtained by placing 500 nodes randomly in a square of side 10 000. The network formed from the edges lying along power optimal routes for $\alpha = 2$ is shown in Figure 2.5. Figure 2.6

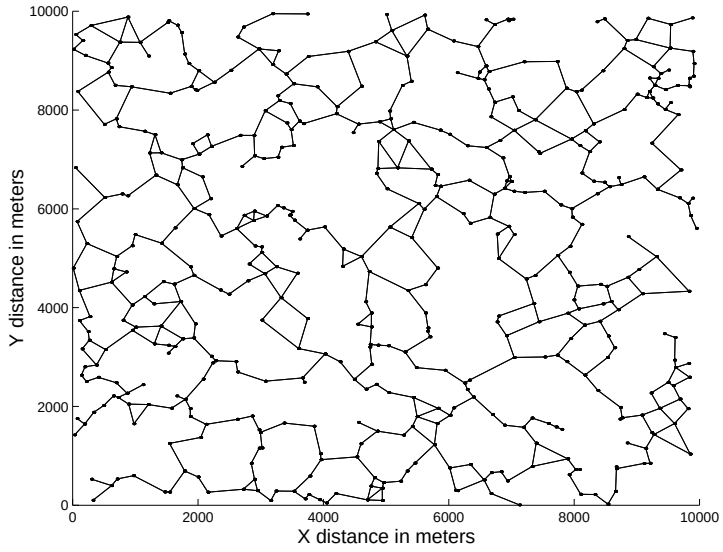



**Figure 2.6** The graph of links lying along power-optimal routes for $\alpha = 4$ is a subgraph of the one for $\alpha = 2$.

shows the corresponding power optimal routes for $\alpha = 4$, which can be seen to be a subgraph of the graph for $\alpha = 2$. Due to the planarity, it can be seen that all hops are only to nearby neighbors; i.e., they use only low power, and power-aware routing, too, prefers many short hops to one long hop. Thus, low power transmission is broadly commensurate with power-aware routing. The actual power consumed: $P_{Tx_{Rad}}(p)$ depends on the efficiency of the RF power amplifier.

(b) *When $P_{Sleep}$ is much less than $P_{Idle}$, then turning the radio off whenever possible becomes an important energy saving strategy.*

This is the case for many commercial-off-the shelf wireless cards. Some estimates in [30, 31] list $P_{Idle}$ to be about 20 times $P_{Sleep}$ for current hardware. "Power management" protocols [32–34], which seek to put nodes to sleep, while maintaining network connectivity and buffering packets for nodes that are "sleeping" become important. However, in this work we do not address the sleeping nodes, but only the awake nodes. Power management schemes can possibly be integrated with our power control schemes to run in unison.

(c) *When a common power level is used throughout the network, then there exists a critical transmission range $r_{crit}$, below which transmissions are sub-optimal with regards*

*to energy consumption.*

The energy consumed for transmitting a packet, using hops of power level $r$, from a source to a destination separated by distance $d$, is given by $\frac{d}{r}(P_{Rx_{elec}} + P_{Tx_{elec}} + cr^{\alpha})$, which is minimized at

$$r_{crit} = \sqrt[\alpha]{\frac{P_{Rx_{elec}} + P_{Tx_{elec}}}{c(\alpha - 1)}} \tag{2.4}$$

To satisfy network connectivity one may need a range even greater than this.

IV. **When the traffic load in the network is high, a lower power level gives lower end-to-end delay, while under low load a higher power gives lower delay.**

At every hop a packet experiences processing delay, propagation delay, and queuing delay. Processing delay includes the time taken by the radio to receive the packet, decode it, and retransmit it if necessary. Propagation delay is the time taken by the radio waves to travel the physical distance separating the nodes. Queuing delay is the time spent by the packets waiting in the queue of the forwarding nodes because the medium is busy. The end-to-end delay for a packet is the sum of the delays it experiences at every hop.

Processing delay grows approximately linearly in the number of hops and is thus inversely proportional to the range. Queuing delay depends on the accessibility of the medium, i.e., on the MAC contention and the interference in the neighborhood. Since contention increases linearly with the range, queuing delay increases superlinearly with the power level, given the convex dependence of delay on load. Power control does not affect the propagation delay much, as it depends only on the end-to-end distance. Thus a higher transmit power implies higher queuing delay, whereas a lower transmit power implies higher processing delay.

Whether the processing delay or the queuing delay dominates depends on the network load. Under low load, queuing delay is insignificant, and thus, it is beneficial to use a higher transmit power which reduces the processing delay. On the other hand, when the network load is high, queuing delay dominates, and it is desirable to use a low transmit power to reduce the total end-to-end delay. This is qualitatively indicated in Figure 2.7. An ideal power control protocol should follow the troughs of these curves.

To verify our predictions and to get an estimate of the cross-over points, we simulated a topology of 80 nodes placed randomly in a 1000 $\times$ 500 m rectangular grid, using the

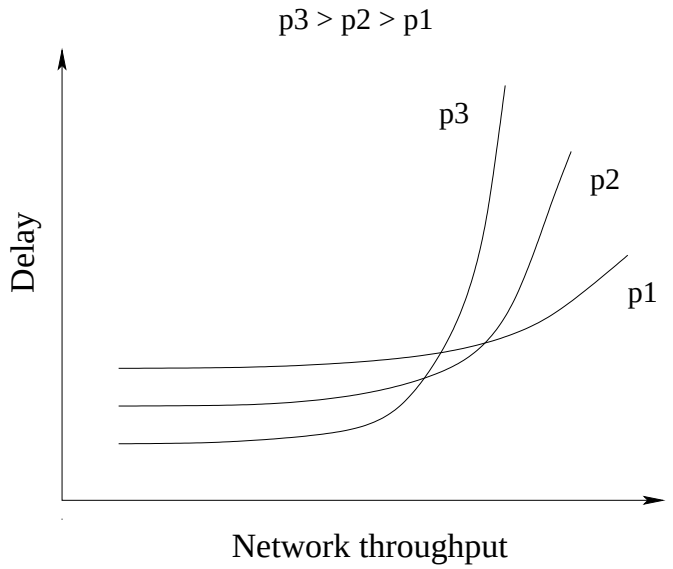



**Figure 2.7** A qualitative sketch of the expected delay-throughput curves at different power levels.

NS2 simulator. The MAC protocol used was IEEE 802.11b and the routing protocol was DSDV. The network load was varied by increasing the number of randomly selected source-destination pairs which carried constant bit-rate UDP traffic. As seen in Figure 2.8, a lower power level can sustain more traffic since it blows up later than a higher power level curve. However at low loads, a higher power level gives a lower delay. The delay jitter also shows a similar trend.

V. **Power control can be regarded as a network layer problem.**

This is a central thesis of our approach to power control. Power control impacts multiple layers of the network stack, including the physical, the data link, and the network layers. Numerous approaches (e.g., [6, 7]) attempt to solve the power control problem at the MAC layer. The strategy is to adjust the transmit power level of every packet such that the SINR at the intended receiver is just enough for decoding the packet. The claim is that this minimizes interference as well as saves energy. One point to note, though, is that the intended receiver is determined by the network layer (i.e., by the routing table entry) and not by the MAC layer. The job of the MAC layer is only to transmit the packet to the receiver specified by the higher layers. Thus, placing power control at the MAC layer does not give the routing protocol the opportunity to determine the optimal next hop or the

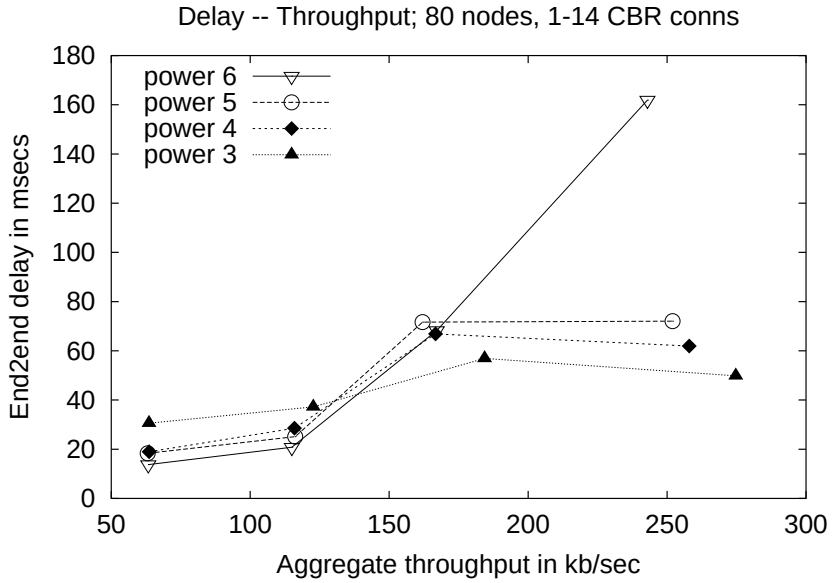



**Figure 2.8** Delay-throughput curves at different power levels obtained through simulations.

intended receiver. In other words, the MAC approach to power control only does a local optimization whereas network layer power control is capable of a global optimization.

When the power level used by a node changes slowly compared to routing updates, power control can be viewed as the "topology control" problem. A more tight coupling of routing and power control can be effected by per-packet power control, which enables us to solve the clustering problem also in a clean way along with the power control problem, as we will see in the sequel. Thus, we argue for power control to be properly situated as a network layer protocol. The above is only a guiding principle. In order to solve the power control problem, we need to show how it can be solved at the network layer resolving all the issues that we have raised. Some solutions may need help from other layers and one may resort to cross-layer design.

## 2.2 The COMPOW Power Control Protocol

A first cut simple solution for power control is the COMPOW protocol [27]. In COMPOW, the goal of the optimization for each node is to (a) choose a common power level, (b) set this power level to the lowest value which keeps the network connected, and (c) keep the energy

consumption close to minimum. A common power throughout the network for all the packets has the key property of ensuring bidirectionality of links due to reciprocity of electromagnetic waves in space, assuming that other factors such as interference are relatively homogeneous. Bidirectionality ensures that the MAC and network layers work properly and also enables use of the standard OSI protocols like ARP, DHCP, etc. We ensure that using too low a power level does not increase the energy consumption by restricting the lowest admissible power level to the one corresponding to $r_{crit}$, in line with the argumentation in Section 2.1, point III(c).

The setting of the lowest common power level is designed to maximize network capacity. In fact, under a homogeneous spatial distribution, it is shown in [22] that choosing a common power level can decrease capacity at most by a factor of $\sqrt{\log n}$, where $n$ is the number of nodes, in comparison to allowing the flexibility of a different power level for each packet at each node.

The next issue is to show *how* this common power level is to be realized at the network level. The architectural solution consists of running multiple independent proactive routing protocols, one at each admissible power level, and a COMPOW agent figures out the lowest common power for connectivity using these routing tables. Various optimizations are further possible. COMPOW has certain appealing features:

1. It provides bidirectional links, assuming homogeneous interference.
2. It allows each layer to function properly.
3. It is theoretically well supported in terms of its design objective of choosing the lowest common power level subject to connectivity.
4. It has a modular implementation at the network layer.

## 2.3 The CLUSTERPOW Power Control Protocol

COMPOW works well if nodes are distributed homogeneously in space, but even a single outlying node could cause every node to use a high power level. So when the spatial distribution of nodes is inhomogeneous, it is obviously not optimal to use a common power level throughout the network. We might allow nodes to use a power level which depends on the

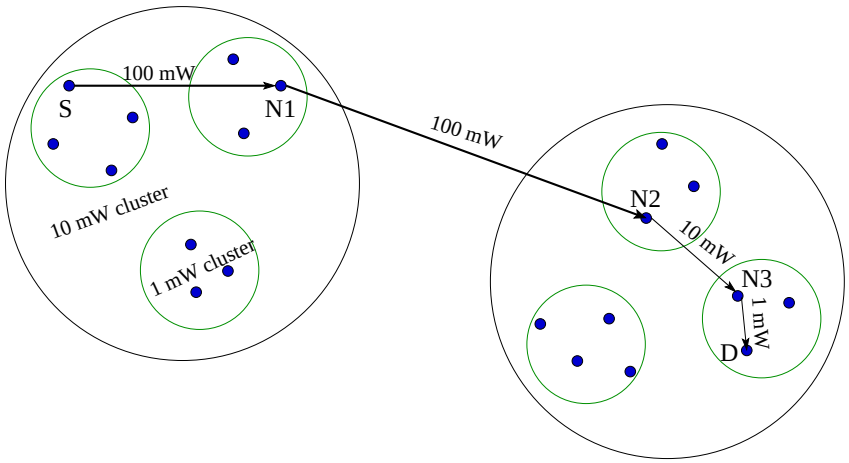



**Figure 2.9** Routing by CLUSTERPOW in a typical nonhomogeneous network.

destination of the packet. This suggests a simple algorithm for routing and power control in clustered networks, which attempts to maximize spatial reuse and hence network capacity. Every node forwards a packet for a destination $d$ using the smallest power level $p$ such that the destination $d$ is reachable, possibly in multiple hops, using only $p$. In some sense this is a greedy algorithm, since every node uses the lowest power level which guarantees reaching the destination according to the information it has. This is executed at the source and every intermediate node. The consequence is that if a node further downstream knows how to reach the destination using a lower power level, then it uses that lower power level for forwarding the packet. Figure 2.9 illustrates the routes chosen, and the power level used when the above algorithm, called CLUSTERPOW [28], is executed on a typical clustered network .

### 2.3.1 CLUSTERPOW architecture and implementation

To implement CLUSTERPOW, each node runs a routing protocol at each power level $p_i$, thereby independently building a routing table $\mathrm{RT}_{p_i}$ by exchanging hello messages at only that power level. To forward a packet for a destination $d$, a node consults the lowest power routing table in which $d$ is present, say $\mathrm{RT}_{p_{min}}$, and forwards the packet at power level $p_{min}$ to the next hop indicated in the routing table $\mathrm{RT}_{p_{min}}$.

The software architectural design of the CLUSTERPOW protocol is similar to that of COMPOW, and is illustrated in Figure 2.10. Each node runs multiple routing daemons, one

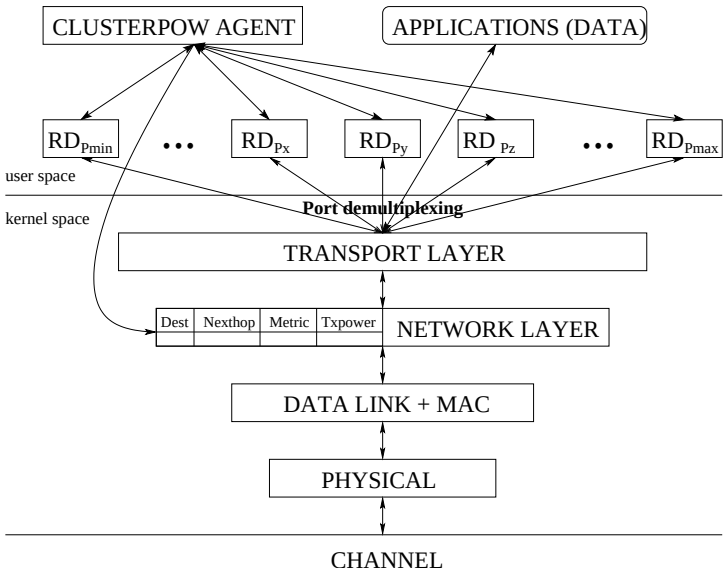



**Figure 2.10** Architectural design of CLUSTERPOW.

for each power level in user-space, and the routing tables constructed are made available to the CLUSTERPOW agent. The agent then populates the entries in the kernel routing table, which is the one actually consulted for forwarding packets. Each entry in the kernel routing table lists not only the next hop for that destination, but also the power level that is to be used for transmission to the next hop.

Consider the network in Figure 2.9. The user space routing tables at each power level and the kernel IP routing table at each of the nodes are shown in Figure 2.11. At node S, the destination D appears (i.e., has a finite metric) only in the 100 mW routing table, with N1 as the next hop. Thus, this entry is copied into the kernel IP routing table and used for routing. The situation is similar for N1, since the destination appears only in the 100 mW routing table, with N2 as the next hop. At N2, however, the lowest power level at which D is reachable is 10 mW. So this is used for routing and the packet is sent to N3, which has D in its 1 mW routing table. Hence the final hop of the packet is at 1 mW. This architecture provides a simple way to implement the CLUSTERPOW algorithm.

The architectural design as presented above capitalizes on certain assumptions about the wireless hardware. The first assumption is that transmission is possible at a small number of discrete power levels, so that we can run a routing daemon at each power level. This is true

1 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | | Inf |

10 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | | Inf |

100 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | N1 | 3 |

Node S

Kernel IP Routing Table

| Dest | NextHop | Metric | TxPower |
|---|---|---|---|
| D | N1 | 3 | 100 mW |

1 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | | Inf |

10 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | | Inf |

100 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | N2 | 3 |

Node N1

Kernel IP Routing Table

| Dest | NextHop | Metric | TxPower |
|---|---|---|---|
| D | N2 | 3 | 100 mW |

1 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | | Inf |

10 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | N3 | 2 |

100 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | D | 1 |

Node N2

Kernel IP Routing Table

| Dest | NextHop | Metric | TxPower |
|---|---|---|---|
| D | N3 | 2 | 10 mW |

1 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | D | 1 |

10 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | D | 1 |

100 mW Routing Table

| Dest | NextHop | Metric |
|---|---|---|
| D | D | 1 |

Node N3

Kernel IP Routing Table

| Dest | NextHop | Metric | TxPower |
|---|---|---|---|
| D | D | 1 | 1 mW |

**Figure 2.11** Routing tables for all power levels, and the kernel IP routing table, at all the nodes in the network of Figure 2.9.

of the current off-the-shelf wireless network interface cards capable of transmit power control. For example, the Cisco Aironet 350 series cards (IEEE 802.11b compliant) allow the transmit power level to be set to one of 1, 5, 20, 30, 50, and 100 mW. The second assumption is that the cards are capable of per-packet power control without much latency. This is of course a prerequisite for any dynamic power control scheme. Unfortunately, this is not entirely true for the Cisco 350 cards that we have used. Even though we could modify the driver to enable per-packet power control, the switching latency is inexplicably as high as 100 ms, when measured at the network layer. We believe that this is a firmware limitation, rather than a fundamental limitation of the electronics, since fast power control is quite common in, e.g., CDMA networks.

Since the firmware on the cards is proprietary we do not have access to it, but we envision that user-demand may lead to the problem being fixed in future.

### 2.3.2 Properties of CLUSTERPOW

1. *CLUSTERPOW provides implicit, adaptive, and distributed clustering based on transmit power.* Clustering based on geographical location could be problematic since GPS, for example, may not work indoors. However, an even more serious problem is that geographical proximity does not guarantee radio proximity, since there may be a radio-opaque obstacle between two nearby nodes. In contrast, in CLUSTERPOW, clustering is implicit and does not require any cluster-head or gateway nodes. The clusters are determined by reachability at a given power level and the clustered structure of the network is respected in the way routes are chosen. The hierarchy of clustering could be as deep as the number of power levels. Clustering is also dynamic and distributed, because it is integrated with a routing protocol that has these properties.

2. *CLUSTERPOW can be used with any routing protocol, reactive or proactive.* In the case of a proactive routing protocol (e.g., DSDV [35]), all the routing tables at different power levels are maintained through hello packets and the kernel routing table is composed from them. For a reactive or on-demand routing protocol like AODV [23], route discovery requests can be sent out at all the power levels available. The lowest power level which results in a successful route discovery can then be used for routing the packet.

3. *CLUSTERPOW is loop free.* The kernel routing table in CLUSTERPOW is a composite of the individual routing tables at different power levels. It is possible that this interaction between routing protocols could lead to packets getting into infinite loops However this is not the case, as we prove below.

**Theorem 1.** *The CLUSTERPOW power control protocol provides loop free routes.*

*Proof.* The proof is based on the key property of CLUSTERPOW, that it chooses routes such that subsequent hops use a sequence of nonincreasing power levels. This is because, when a particular power level $p$ is used, the destination is present in the routing table corresponding

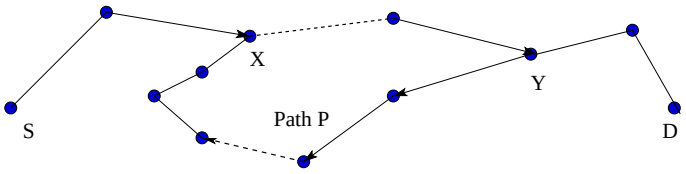



**Figure 2.12** Suppose there is a loop on the path P from S to D. Dashed lines indicate paths consisting of many hops.

to $p$, and there is guaranteed to exist a path of power level at most $p$ from the current node to the destination. Thus, further downstream, the power can only decrease. Thus, if there is a loop as shown in Figure 2.12 (i.e., a packet on its way from node S to node D follows the path S-X-Y-X...), then all the hops on the loop have to be of the same power level. But that cannot happen since the underlying routing protocol is loop free. □

### 2.3.3 CLUSTERPOW implementation and software architecture

CLUSTERPOW has been implemented for Linux. We now describe the software architecture (see Figure 2.13) and the implementation details. The first task is to run multiple routing daemons at different power levels. In Linux, route discovery and maintenance is done by user space programs called routing daemons, and the actual packet forwarding is done by consulting the kernel IP routing table, which is populated by the routing daemons (see [36]). Thus, running multiple routing daemons simply involves starting many of these routing daemons, one for each power level, on preassigned ports. They use UDP packets for communication, thus transport layer port demultiplexing ensures that a routing daemon at a particular power level communicates only with its peers at other nodes. From the routing tables at all the power levels, the composition of the kernel routing table is done by the CLUSTERPOW agent running in user-space. The routing daemons themselves do not modify the kernel routing table directly.

The next task in the implementation is to make the Linux kernel aware of the concept of transmit power of a packet. The most natural way to do this is to add a field (`txpower`) to the data structure associated with a packet, called `skb`, which is of type `struct skbuff`. The structure `skb` contains various protocol headers and other layer independent parameters, apart

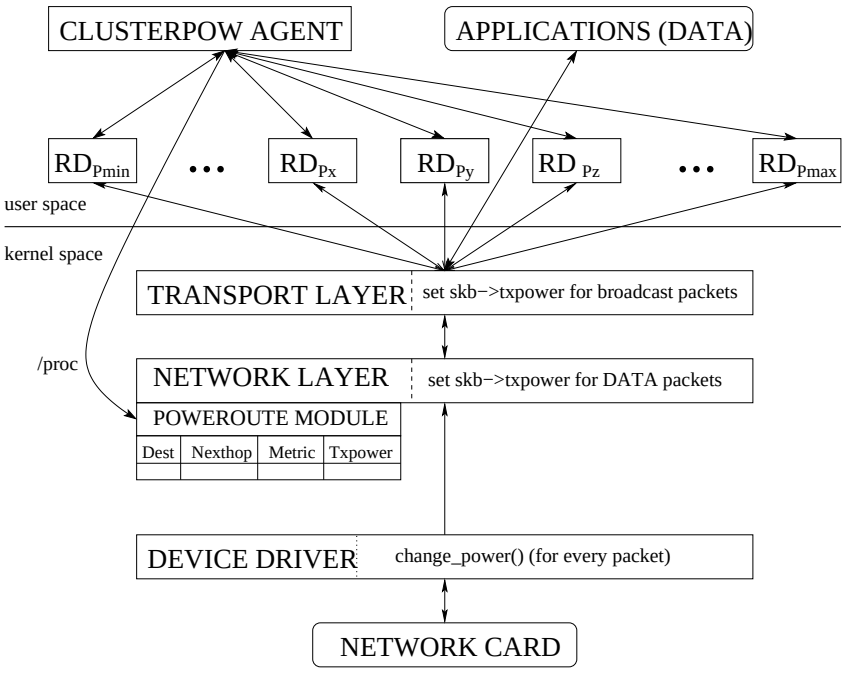



**Figure 2.13** The software architecture of CLUSTERPOW.

from the data payload. As we will see, `skb→txpower` is set by the network layer and used by the device driver to set the power on the card before transmitting the packet on the air.

Now we consider the issue of extending the kernel routing table by adding an extra field `skb→txpower`, which specifies the power level to be used when forwarding packets for that destination. A possible approach is to modify the core Linux IP forwarding code to add this field to the kernel routing table and to modify the forwarding functionality accordingly. Not only is this task tedious, but it also has the disadvantage of requiring extensive changes in the kernel core, thereby making it unacceptable for possible inclusion in the standard distribution. This modification would also necessitate a change in the kernel API and would break programs like "route" which rely on this API.

The approach we take is to implement the addendum to the kernel routing table (i.e., the txpower field) in a kernel module called *poweroute*. This modules uses Netfilter, a generic packet filtering and mangling framework in the Linux 2.4 kernel, to intercept packets after they have consulted the kernel routing table and sets the `skb→txpower` field in accordance with the additional table, which is administered by user-space programs using the `/proc` filesystem.

The transmit power of broadcast packets (e.g., hello packets from the routing daemon) cannot be decided by the routing table, since different broadcast packets may need to be sent at different power levels. Hence, the transmit power for such packets has to be specified by the application sending these packets. For hello packets this application will be the routing daemons running at different power levels. We have provided such a mechanism by modifying the `sendto()` system call, so that the transmit power can be specified by using certain values for the flags argument of this call.

The network device driver was modified so that it can read the transmit power from the `skb` and set it on the card. The driver also maintains a new variable called DefaultTxPower, which is used to transmit packets for which no power level has been specified from above. This is used by the COMPOW agent to specify the default node power level, whereas for CLUSTERPOW it is set to the max power level. We have used the Cisco Aironet 350 cards in our implementation, which are the only commercially off-the shelf available cards supporting multiple transmit power levels.

The implementation described above is architecturally clean and modular. It makes minimum intrusions into the kernel; most of them being concerned with making the kernel aware of the concept of a per-packet transmit power. The implementation is fairly generic and could be used for other power control protocols. COMPOW and MINPOW (described later) use the same architecture. Source code for the Linux 2.4.18 kernel is available online [37].

## 2.4 Recursive Lookup Schemes

While CLUSTERPOW attempts to maximize spatial reuse, it does not achieve optimality in that regard. We can do better as demonstrated in Figure 2.14, where the earlier 100 mW hop from node S to node N1 is now replaced by shorter hops. A possible scheme to achieve the routing shown in Figure 2.14 is recursive lookup of routing tables. In this scheme the next hop is recursively looked up in successively lower power level routing tables, until we get to the lowest power level routing table at which the next hop is reachable. Thus, in Figure 2.14, the next hop N1 at node S is looked up in lower power routing tables to find that it is reachable at

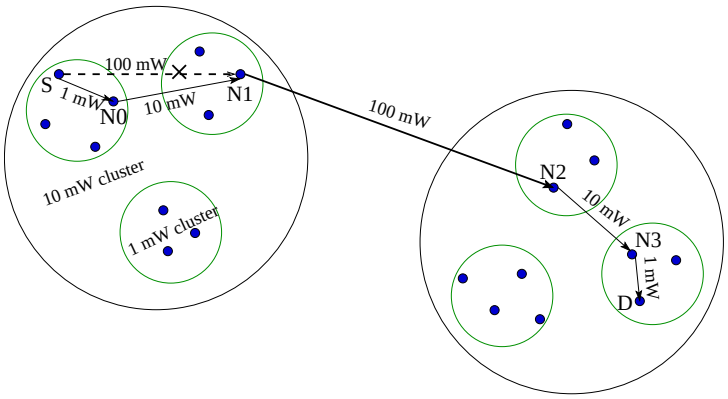



**Figure 2.14** Modifying the CLUSTERPOW protocol, so that the 100 mW hop from S to N1 can be replaced by two hops of 1 mW and 10 mW each.

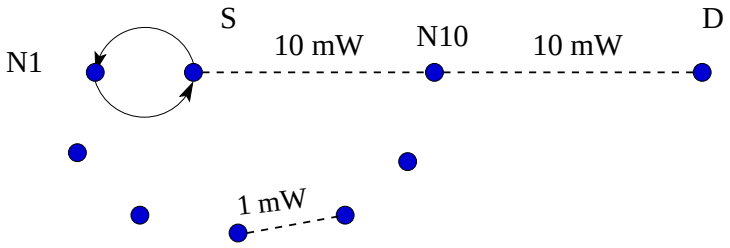



**Figure 2.15** The recursive lookup scheme is not free of infinite loops.

10 mW through N0, which in turn is reachable at 1 mW. So ultimately the packet is given to N0 at 1 mW.

This recursive lookup scheme seems to have achieved a finer optimization with regard to network capacity, but it does not guarantee freedom from loops, i.e., packets can keep coming back to a node indefinitely. The network in Figure 2.15 provides a counterexample. Node S needs to send a packet to node D. It determines that the next hop is node N10 in the 10 mW routing table. Recursive lookup for N10 reveals that it is reachable at 1 mW, with next hop is N1. Thus S forwards the packet to N1 at 1 mW. After the packet reaches N1, it runs the same algorithm. It finds that the lowest power level at which D is reachable is 10 mW and that the next hop is S. Since S itself is reachable at 1 mW, the packet is handed over back to node S, and we have an infinite loop.

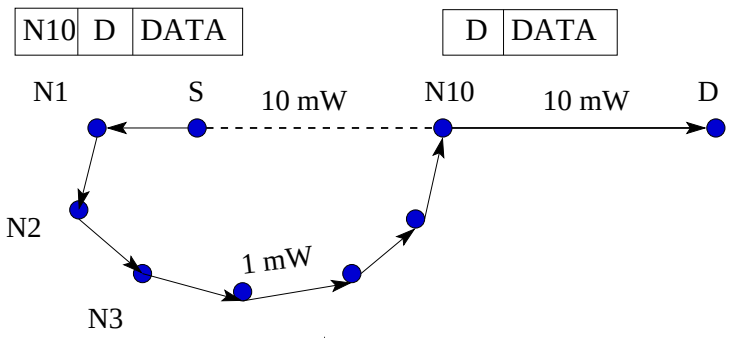



**Figure 2.16** Tunneled CLUSTERPOW protocol resolves the infinite routing loop of the network in Figure 2.15. The headers added to the packet, as it travels along the route, are also shown.

### 2.4.1 The Tunneled CLUSTERPOW protocol

To ensure that the packets in the recursive lookup scheme do reach their destination, we need to ensure that they make progress towards the destination. One way to achieve is this is to *tunnel* the packet to the next hop using hops of lower power level, instead of sending it directly. Tunneling can be viewed as a way of introducing some state or memory in the packet, and can be achieved by IP in IP encapsulation. While doing a recursive lookup for the next hop, we also recursively encapsulate the packet with the address of the node for which the recursive lookup is being done. The decapsulation is also done recursively when the packet reaches the corresponding next hop. This gives rise to the Tunneled CLUSTERPOW protocol.

As shown in Figure 2.16, this does resolve the loop in the example of Figure 2.15. Now when node S forwards the packet to N1, it encapsulates the packet with the address of N10. Thus N1 does a routing lookup, not for the destination D, but for node N10. It finds that N10 is reachable at 1 mW through the path N2, N3 . . . , and it forwards the packet to N2 at 1 mW. When the packet gets to N10 it decapsulates the packet, and then sends it to D at 10 mW. Thus, the packet does reach its destination. This is true in general, as we now prove.

**Theorem 2.** *Tunneled CLUSTERPOW ensures that packets reach their destinations.*

*Proof.* Suppose the transmit power levels are indexed from 1 through $t$, ordered such that power level $t$ is the lowest. We provide a proof by induction on the number of transmit power levels $t$. The base case $t = 1$ is obvious, since it reduces to a single routing daemon for a fixed power

level, and the underlying routing protocol is assumed loop free. Assume that the Tunneled CLUSTERPOW protocol provides routes free of infinite loops when $t-1$ power levels are in use. This is the induction hypothesis. Now we add the $t^{th}$ power level, which is lower than any power level already in use. Here we note that Tunneled CLUSTERPOW is a refinement to CLUSTERPOW, as seen in Figure 2.14. If a packet from source S to destination D visits the sequence of nodes $\{a_i\}$ in CLUSTERPOW, and the sequence of nodes $\{b_i\}$ in Tunneled CLUSTERPOW, then $\{a_i\}$ is a subsequence of $\{b_i\}$. This is ensured by the encapsulation or the tunneling mechanism. Thus, if a packet in Tunneled CLUSTERPOW can get from a node $a_j$ to node $a_{j+1}$, for any $j$, then it will indeed get to its destination by Theorem 1, since CLUSTERPOW is loop free.

Therefore, consider the subproblem of getting from node $a_j$ to node $a_{j+1}$, for any $j$. Suppose CLUSTERPOW was using a power level $p$ in getting from node $a_j$ to node $a_{j+1}$. Tunneled CLUSTERPOW will introduce more hops between $a_j$ and $a_{j+1}$ only if a power level strictly less than $p$ is used. This subproblem thus reduces to running the Tunneled CLUSTERPOW protocol with $t-1$ power levels, which is free of infinite loops by the induction hypothesis. □

### 2.4.2 Architecture and implementation issues

Implementing Tunneled CLUSTERPOW is more complicated because of the dynamic recursive encapsulation and decapsulation, increased forwarding overhead due to the increased size of the IP header, and increased processing times. Due to these these issues, we have not implemented this protocol. Nevertheless, it is a concrete example of the sort of schemes that are possible with a sophisticated composition of various individual routing tables built at different power levels.

## 2.5 MINPOW Routing and Power Control Protocol

So far we have attempted to optimize network capacity by increasing spatial reuse. Energy consumption is however also an important metric, and as we saw in Section 2.1, network capacity and energy consumption are not optimized simultaneously for current off-the-shelf wireless cards. We present another protocol, called MINPOW, which globally optimizes the

total energy consumption. It is essentially distributed Bellman-Ford with energy consumption as the metric. The basic idea has been proposed previously in various forms [10, 38–40], with a variety of metrics like signal strength, transmit power cost of the link, a node's remaining battery life, or variance in battery life among all nodes. However, there has been no actual implementation, possibly because many of these schemes require support from the physical layer. Our contribution involves an architecturally clean implementation in Linux based on clearly identifying the various components of energy consumption, and estimating them without assuming any physical layer support.

Of the three components of links cost (elaborated in Section III), $P_{Tx_{elec}}$ and $P_{Rx_{elec}}$ are known locally to the transmitter and receiver respectively, while $P_{Tx_{Rad}}(p)$ can be calculated if the smallest transmit power $p_l$ required to traverse a link $l$ can be estimated. The term $p_l$ cannot be accurately estimated by assuming a path loss model for the channel, since the parameters of the path loss model depend on the environment and can vary significantly. Moreover, distance measurement requires nodes to be equipped with location measurement equipment and even then, it may not accurate because of obstacles in the environment. We circumvent these problems by estimating the link cost using control packets at the network layer. This mechanism is robust to channel models and fluctuations, does not require any physical layer support or location measurement, and naturally takes advantage of the discreteness of power levels.

### 2.5.1 MINPOW implementation

We have modified the DSDV implementation in [19] to implement MINPOW. To estimate the link cost, every node proactively sends hello packets at each of the transmit power levels available, all of them containing the same sequence number. Only the hello packets at the maximum power level contain the routing updates. The rest are only "beacons" which contain the address of the originator, the total power consumed $P_{Tx_{total}}$ in transmitting that packet, the transmit power level $p$ used for transmitting the packet, and the sequence number of the corresponding maximum power level hello packet. Note that $P_{Tx_{total}} = P_{Tx_{elec}} + P_{Tx_{Rad}}(p)$ where $p$ is the transmit power level of the current beacon packet. The neighbors receiving these beacons set the link cost to be the minimum $P_{Tx_{total}}$ value among the beacons that they successfully

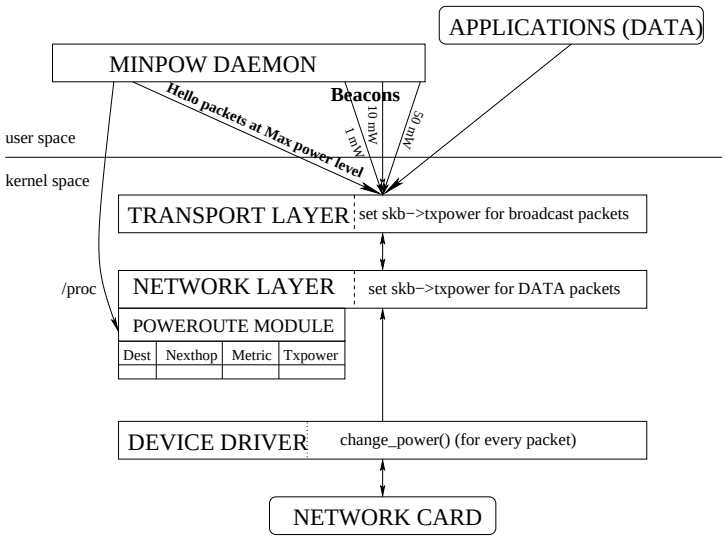



**Figure 2.17** The MINPOW software architecture.

received, plus the energy they spent in receiving, i.e., link cost = $\min_{beacons}(P_{Tx_{total}}) + P_{Rx_{elec}}$. This link cost is then used in the distance vector algorithm for computing the routes. The corresponding transmit power $p$ of the beacon which achieved the min is used for sending packets to the next hop. The software architecture for this MINPOW implementation is illustrated in Figure 2.17.

The method suggested above works for both proactive as well as reactive routing protocols. Most reactive routing protocols, e.g., AODV [23], use beacons for sensing link status, i.e., to check if a neighbor has moved away. These beacons can be sent at all available power levels in turn, and can be used to estimate the link cost as described above. The route requests themselves are sent at maximum power, but the nodes use the link cost as calculated above.

### 2.5.2 MINPOW properties

To summarize, the MINPOW protocol has the following properties:

1. It provides a globally optimal solution with respect to total power consumption. This may not be the optimal solution for network capacity, since, in general, the two objectives are not simultaneously satisfiable.

2. MINPOW provides loop free routes. This is true because the distributed Bellman-Ford algorithm with sequence numbers is loop free for non-negative link cost.

3. No location information or measurement support from the physical layer is needed.

4. The architecture works for both proactive, as well as reactive routing protocols.

## 2.6 Power Control Adaptive to Traffic Load

The schemes considered above maximize network capacity by increasing spatial reuse. However, end-to-end delay is also an important metric. We have seen in Section 2.1 that using a higher transmit power can reduce delay when the network load is low. We capitalize on this in designing the LOADPOW power control protocol which adapts the transmit power according to the network load. It opportunistically uses a higher transmit power level whenever the network load is low, and lowers the transmit power as the load increases. The LOADPOW algorithm attempts to avoid interference with ongoing traffic by making each node refrain from using a transmit power that would interfere with an ongoing communication in the neighborhood. This can be realized by modifying IEEE 802.11's notion of NAV, which every node uses to keep track of the time until which the medium around it is busy, and until which it is forbidden to transmit. We capitalize on the fact that a node whose NAV is marked busy may be able to transmit at a lower power level without disturbing ongoing communications. We propose to modify the NAV mechanism so that every node, say $a$, also dynamically keeps track of the list of current nodes, *busy_list*, which cause it to remain silent, i.e., nodes which are currently participating in transmissions which interfere with $a$. The forwarding decision for a packet is made by the MAC just before transmitting the packet, by making a call to the LOADPOW agent which is the routing agent on the node. For each node $b_i$ in the *busy_list*, the LOADPOW agent finds, by looking in the various routing tables, the highest power level at which $b$ is not reachable, i.e., which does not interfere with $b$. The min of this power level over all elements in the *busy_list* gives the *safe power level* for $a$. It denotes the power level at which $a$ can transmit without "disturbing" any ongoing communication. Forwarding is done by consulting the routing table corresponding to *safe power level*. A similar procedure is followed for the CTS.

From the perspective of a node, when the network load is low, the medium around it will be busy less often and it shall be able to use a higher transmit power more of the time. As the load increases, there will be on average more communications nearby, and the node will use a lower power so that it does not interfere with those communications. Note that our protocol involves cross-layer interaction between the MAC and the network layer. It is based on the multiple routing table architecture presented earlier but does not rely on channel estimation or position information.

It should be noted that LOADPOW may have temporary routing loops, since at each hop a different power level routing table may be consulted. But if the network load reduces the packets should reach their destination comfortably. The temporary loops may be a generic issue with any opportunistic, distributed load based protocol. A second issue is related to practical implementation. We have assumed that the MAC can make a function call to the forwarding functionality, possibly through a call_back function pointer which is put in the packet when forwarding. Similarly, a call to the ARP cache may also be needed. This may be difficult to do in a real operating system. Another subtlety is that the forwarding decision is actually made before sending the RTS, so that the RTS can be sent to the next hop which has been decided by the LOADPOW agent. However the DATA packet is sent only after the CTS is received. The $busy\_list$ could change in the meantime and possibly invalidate our forwarding decision. However, it can change only for the better, i.e., the $safe\_power\_level$ can only increase because all nodes who hear the RTS are required to remain silent for a duration which allows the CTS reply to be received. Finally, we note that the LOADPOW cross-layer scheme leads way to a MAC protocol which can work with heterogeneous power levels.

## 2.7 Experimentation

Based on the actual implementation, we have verified the correct formation of the routing tables as predicted by the COMPOW, CLUSTERPOW and MINPOW algorithms, and have successfully conducted tests with several topologies on our ad hoc networking testbed. However our goal of testing to quantify throughput and delay measurements for these topologies proved infeasible due to the nonsuitability of the Cisco wireless cards for per packet power control (the

**Table 2.1** Relevant simulation parameters.

| | | |
|---|---|---|
| Simulator | NS2 version 2.26 | |
| | Simulation time | 1000 s |
| | Traffic | TCP and constant bit rate UDP |
| MAC | IEEE 802.11 | |
| | Link data rate | 2 Mb/s (basic rate = data rate) |
| | Number of transmit power levels | 6 |
| | Ranges corresponding to the power levels | 50, 90, 130, 170, 210, 250 m |
| | Carrier Sensing range | Same as communication range |
| Routing protocol | DSDV | |
| | Periodic update interval | 15 s |
| | Settling time | 6 s |
| | Min update period | 3 s |

only ones capable of power control, to our knowledge, at the time of this writing). Not only was the power switching latency very high, but frequent power level changes caused these cards to crash. Thus, any actual experimentation with a non-significant amount of traffic proved impossible.

## 2.8 Simulations

To get some quantitative estimate of performance, we resorted to simulations. COMPOW and CLUSTERPOW were implemented in the NS2 simulator, closely following the corresponding implementation architecture in Linux. The modified NS2 source code, tcl scripts and the scenario files are available online [37]. The simulation parameters are listed in detail in Table 2.1. The interference range of IEEE 802.11 was made equal to the carrier sensing range to enable us to study the effect of power control in isolation.

### 2.8.1 An example topology

We first present results for a specific topology which clearly demonstrates the benefits of using an appropriately low power level. The topology is specifically chosen because it avoids the so called "relaying penalty," which is incurred due to the silencing of nodes within range of the transmitter and the receiver when packets are relayed. As shown in Figure 2.18, only

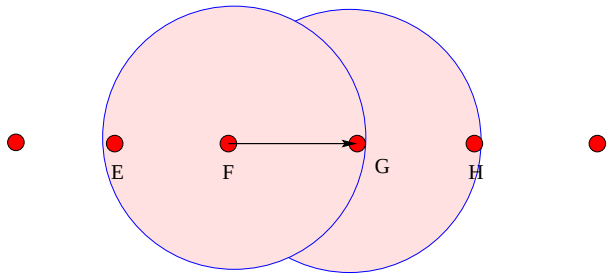



**Figure 2.18** Illustrating the "relaying penalty" in a linear network.

one of 3 hops E-F, F-G, or G-H can be active at a time. For a direct transmission using higher power, there is no relaying penalty.

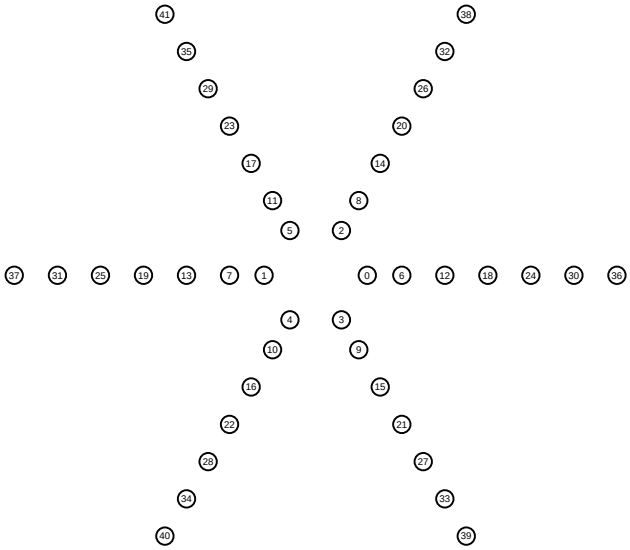



**Figure 2.19** A hexagonal topology.

Relaying penalty is a constant factor per flow and becomes insignificant if the number of hops per flow is greater than a small constant (4 or 5 in case of IEEE 802.11). However, we are constrained by the capability of the NS2 simulator to simulate wireless networks large enough to ameliorate the relaying penalty as well as demonstrate spatial reuse. The network shown in Figure 2.19, with flows along each arm of the hexagon, avoids the relaying penalty by ensuring at least 3 hops at all the power levels used, and yet gains from spatial reuse. The results for running six TCP flows along the six arms of the hexagon (Nodes 36→0, 37→1, 38→2,

39→3, 40→4 and 41→5) are shown in Table 2.2. CLUSTERPOW achieves a higher aggregate throughput than both COMPOW and DSDV running at power level 3.

**Table 2.2** TCP traffic on a hexagonal topology.

| Protocol | Agg. Throughput | Average Delay | St. Deviation of Delay | Routing Overhead |
|---|---|---|---|---|
| CLUSTERPOW | 790.4 kb/s | 59.6 ms | 16.5 ms | 119.4 kb/s |
| COMPOW | 604.2 kb/s | 52.0 ms | 16.2 ms | 96.2 kb/s |
| $DSDV_3$ | 545.1 kb/s | 39.0 ms | 16.1 ms | 18.9 kb/s |

### 2.8.2 Communicating with isolated nodes

Now we consider a case where the advantage of using CLUSTERPOW is clear. It consists of a cluster and a single outlying node as shown in Figure 2.20. Both COMPOW and $DSDV_6$ (DSDV running at power level 6) use a high power for all communications, whereas CLUSTERPOW uses the max power level only when the outlying node (node 30) is involved in a communication. Twelve TCP sessions were run for this simulation, one of which involved the outlying node 30. The results shown in Table 2.3 indicate that CLUSTERPOW can achieve higher throughput at lower delays.

**Table 2.3** TCP traffic for the single outlying node case.

| Protocol | Agg. Throughput | Average Delay | St. Deviation of Delay | Routing Overhead |
|---|---|---|---|---|
| CLUSTERPOW | 4875.3 kb/s | 23.2 ms | 12.4 ms | 35.6 kb/s |
| COMPOW | 3204.9 kb/s | 159.8 ms | 905.6 ms | 38.8 kb/s |
| $DSDV_6$ | 3188.3 kb/s | 2326.6 ms | 2643.1 ms | 9.1 kb/s |

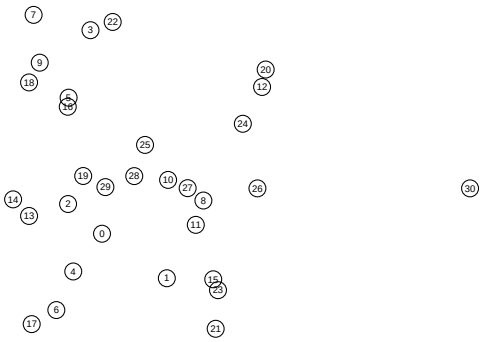



**Figure 2.20** Single outlying node.

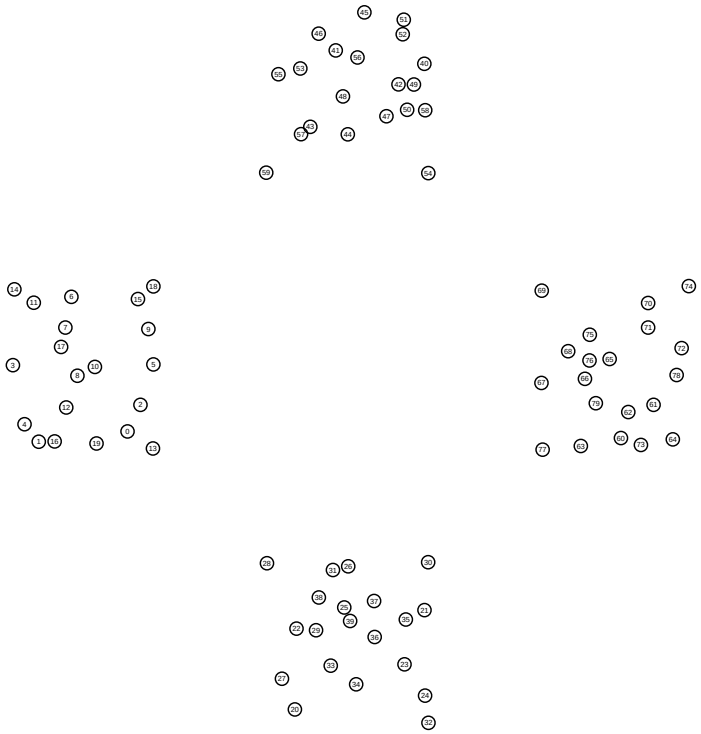


**Figure 2.21** A clustered topology.

**Table 2.4** TCP traffic on a clustered topology with mostly 1-hop connections.

| Protocol | Agg. Throughput | Average Delay | St. Deviation of Delay | Routing Overhead |
|---|---|---|---|---|
| CLUSTERPOW | 18719.9 kb/s | 135.4 ms | 90.5 ms | 75.0 kb/s |
| COMPOW | 7954.2 kb/s | 3781.6 ms | 3652.0 ms | 76.4 kb/s |
| $DSDV_6$ | 7657.2 kb/s | 1244.7 ms | 1423.7 ms | 45.2 kb/s |

### 2.8.3 Clusters

We next simulated a topology consisting of four clusters as shown in Figure 2.21. The advantages of using a lower power level are most evident when most of the traffic is intracluster allowing spatial reuse. We simulate two traffic patterns: TCP and constant bit rate UDP. For the TCP scenario, each cluster has six single hop intracluster connections, and one intercluster connection. The intracluster connections were chosen to allow spatial reuse. The results in Table 2.4, averaged over 6 different random topologies with similar clustered structure, clearly demonstrate the benefit of using power control in such a scenario. The same simulations were then repeated with constant bit rate UDP traffic instead of TCP. The sending rate was the same for all flows and was varied from 50 kb/s to 140 kb/s over different simulation runs. The

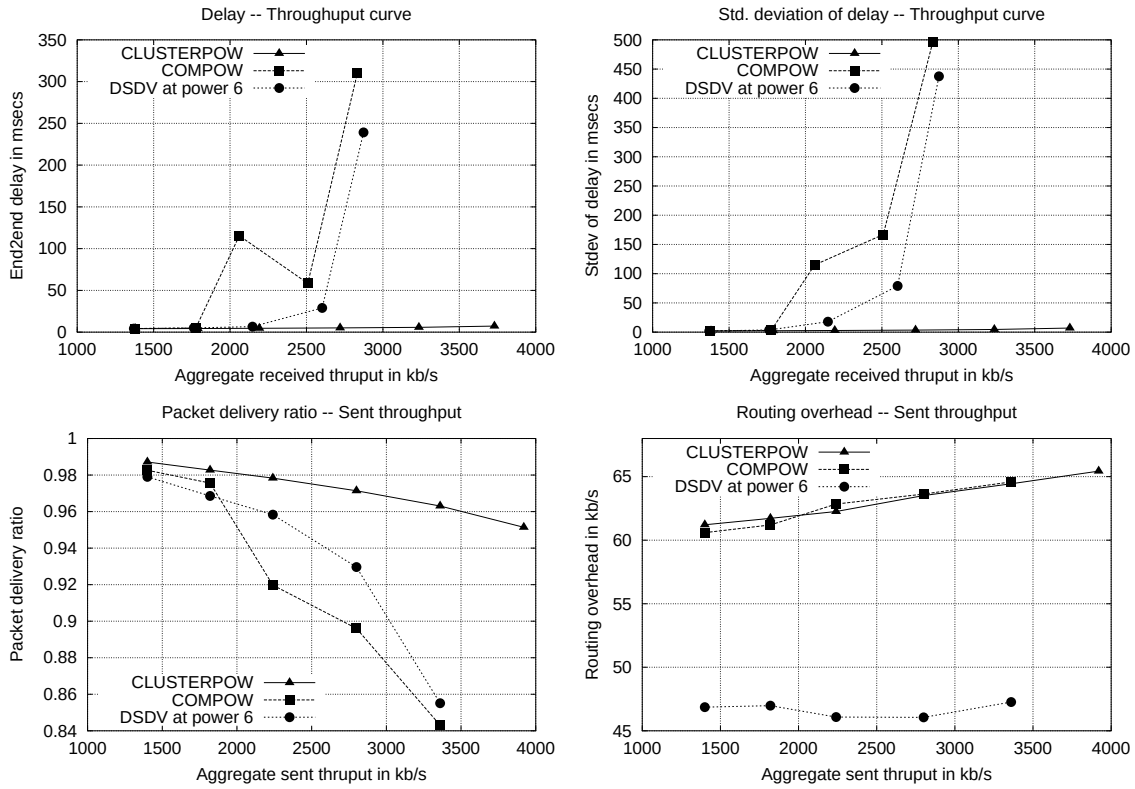



**Figure 2.22** Constant bit rate UDP traffic in a clustered topology of 80 nodes, with carefully selected one-hop 24 intracluster and 4 intercluster connections.

results shown in Figure 2.22 indicate that CLUSTERPOW can provide lower delay and lower delay jitter while ensuring a higher packet delivery ratio (total received throughput/total sent throughput).

For the second scenario (in the same topology Figure 2.21), we randomly select the source destination pairs but with the constraint that 80% of the traffic is intracluster, noting that long distance communication is expensive in ad hoc networks [22]. We randomly picked four intracluster source destination pairs in each cluster, and four intercluster source destination pairs for the network. We send constant bit rate (CBR) UDP traffic between these source destination pairs, where the rate for each flow is varied from 30 kb/s to 200 kb/s over different simulation runs. As seen in Figure 2.23, CLUSTERPOW gives a low delay as well as low delay jitter, until a much higher network load compared to COMPOW and $\text{DSDV}_6$ running at max power level, while packet delivery ratios are similar.

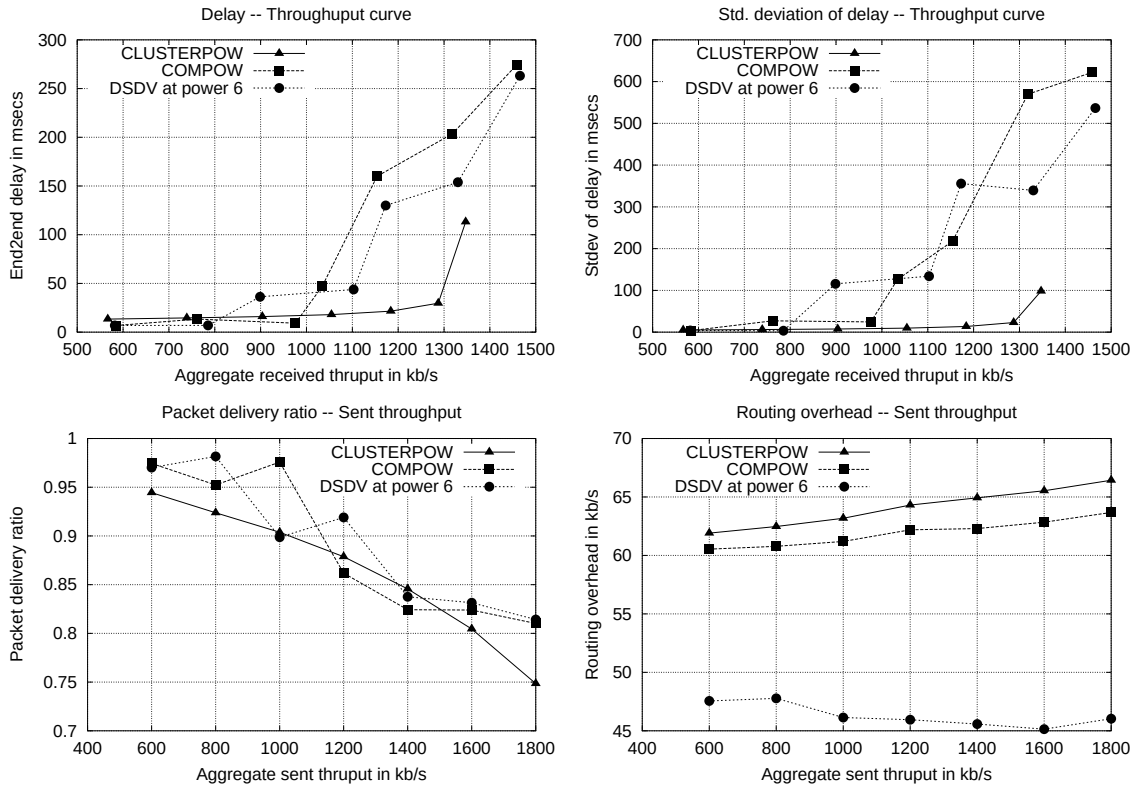



**Figure 2.23** Constant bit rate UDP traffic in a clustered topology of 80 nodes, with randomly selected 16 intracluster and 4 intercluster connections.

### 2.8.4 Comments

Our simulations are by no means a thorough performance evaluation. At best they constitute quantitative estimates for the performance of the protocols in some scenarios. The physical model of interference in NS2 is simplistic, accounting for only one interferer at a time when calculating the SINR. This does not provide an accurate estimate of the performance of the MAC layer. We are also limited by the scale of the system that can be simulated. Another issue was the effort needed to retrofit our clean implementation architecture in Linux to the simulator environments. For definitive assessments, we need to be careful in isolating the effects of MAC and transport layers, which have their own plethora of problems for ad hoc networks. Final performance evaluations will probably have to wait the availability of wireless hardware capable of per packet power control, and the emergence of a complete network stack designed for ad hoc networks, especially the MAC and transport layers.

### 2.8.5 NS2 implementation details

In this section we provide some details on the modifications of the NS2 simulator to implement the power control protocols. We wanted our NS2 implementation to be as close to the real implementation as possible. For this significant changes were needed to the NS2 simulator. Our implementation was done in version 2.26 of the NS2 simulator.

The power level of a packet in NS is determined by the `Pt_` variable which is a member of the class `Phy/WirelessPhy` defined in the file mac/wireless-phy.h. The power level can be changed to x Watts for all nodes globally in tcl through the command: `Phy/WirelessPhy set Pt_ x`. To set the power level of different nodes to different values one has to access the WirelessPhy class from a node's array of interfaces (`netif_`) and then set the corresponding `Pt_` to the desired value. For example, the following command sets the power level of node node_(0) to 0.5 W.

```
[[lindex [$node_(0) array get netif_] 1] set Pt_ 0.5]
```

However we need per packet power control, not just per node power control. Any per packet processing in NS should be done in C++ as it is extremely inefficient to do it in Tcl. For per packet power control, transmit power needs to be associated with every packet. An appropriate place for this is `hdr_cmn` in the file common/packet.h, which is somewhat analogous to `struct skbuff` in the Linux kernel. We add a new field `txpower_` in `hdr_cmn`, which is an integer denoting the power level. Code is added in `WirelessPhy::sendDown()` to extract the power level from the common header and set `Pt_` equal to the transmit power corresponding to that power level. The mapping between power levels and the exact value in watts is controlled by the variables `TxPowerVal_x_` (where $x$ goes from 1 to 6), which can be set from tcl space. Another variable `DefaultTxPowerLevel_` can be set from Tcl for every node which is used for setting the `Pt_` for those packets in which `txpower_` in `hdr_cmn` is set to zero.

The field `txpower_` in `hdr_cmn` should be filled for most packets by a routing functionality analogous to the kernel IP routing table in Linux. There is no such central routing table in NS, at least for the mobile ad hoc routing protocols like DSR, AODV, etc. They are implemented as "agents." They not only construct the routing tables but also do the job of forwarding by directly modifying the IP header in the packet. This is architecturally unclean as it destroys the

routing-forwarding separation [36] present in most operating systems. This presents a problem for us, as our design critically depends on running multiple routing daemons and having a master routing table which is used for forwarding. We resolved the problem by writing a CLUSTERPOW routing agent which actually does not exchange any routing hello packets, rather just maintains a forwarding table (along with the txpower field for each destination) and fills it in the IP and common header. This is similar to hardwired or fixed routing, and this agent is attached to port 255. We will hence forth call it the CLUSTERPOW forwarding agent. Similarly, we also have a COMPOW forwarding agent. The appropriate agent can be selected through a node's adhocRouting attribute. The CLUSTERPOW routing table is filled by other routing agents which are attached to ports 101 through 106. This attachment is done in our tcl script for the simulation. Agents on a node correspond to daemons in an OS which would communicate through an inter-process mechanism. The NS2 analogue for inter-process communication is function calls on the appropriate object.

We have used the DSDV agent in our simulations, modified to run on a specified port at the specified power level. One would think that running multiple routing agents in NS would be just a matter of instantiating more DSDVAgents. However, in NS, broadcast packets would not be communicated between agents unless they were bound to the special port 255. This is because all broadcast packets come to the routing agent, which does not know what to do with them as they are not explicitly present in the routing table. In NS the routing agent should give the broadcast packets to the node's port demultiplexer which would give it to the correct agent. However the port demultiplexer was not always correctly attached to the routing agent, in the file tcl/lib/ns-lib.tcl. Things worked as expected after this bug was identified and fixed. The `recv()` function in the forwarding agent (CLUSTERPOW or COMPOW) also had to be changed so that broadcast packets are correctly handed over to the port demultiplexer.

We have modified the transmit power at which the packets will be sent, and that is all one has to do in a testbed in an actual implementation. Once a packet is given to a wireless card, it sends out all the MAC control packets RTS automatically at that power level. But in NS the power level for each of those packets has to be specified explicitly. Otherwise the RTS packets would still go out at the default maximum power level, thus providing no spatial reuse from power control. We have made these changes in the file mac/mac-802_11.cc. We have also

changed the code so that the CTS and ACK packets are sent back at the same power level at which the corresponding RTS packet was sent from a neighbor node. The power level of the RTS packet is put in the packet header.

We have not bothered to modify the NS code for tracing packets to reflect the new power level information. Nor have we modified the code for the Networm Animator (NAM). In fact we do not generate any trace files at all. All the computation for throughput, delay, delay jitter and routing overhead is done in the respective agents along with the simulation and the values for each flow are printed at the end. This saves us the trouble of dealing with the extremely huge trace files which also slow down the simulations considerably.

## 2.9 Related Work

Most work on power control in ad hoc networks considers the problem as either topology control, an energy optimization problem, or a MAC layer problem. Topology control associates with each node a power level, which varies with time (if at all) at a time scale slower than that of routing updates. However, we allow for per packet power control, which is more general than topology control. Examples of topology control include [8, 9, 41, 42], which control the node power based on the number of hop neighbors, end-to-end throughput, or direction information.

A lot of work on power control primarily concerns itself with optimizing energy consumption. For example, power management schemes such as [32], Span [33], Geographical Adaptive Fidelity (GAF) [34] etc. devise sleep and wakeup schedules for nodes. Some of these schemes can be implemented in conjunction with some of protocols that we have present. There are other schemes which can be generically called "power-aware" routing. Like MINPOW, they are a variation of distributed Bellman-Ford with a power based metric. Some examples of work in this category include [10, 38–40]. More recently, [43] and [44] have done important work in energy efficient routing. There has also been significant work in the area of energy efficient broadcast and multicast in the wireless medium. These include [45–47].

Power control is often considered a problem belonging completely at the MAC layer, thus MAC protocols dealing with power control have been proposed. PCMA [48], PAMAS [49],

and [50, 51] are some examples. Some of these require multiple channels, which may not be practical. A power control scheme adaptive to network load has been presented in [52].

## 2.10 Concluding Remarks

Power control is a prototypical example of a cross-layer design problem. We have identified the impact of power control on a variety of parameters and phenomena, and presented fundamental design principles. We have developed protocols guided by these principles, taking into account architectural considerations for implementing them in an actual system. Some of the protocols have been implemented and tested. Perhaps, the holistic approach used here may be useful in other contexts.

# CHAPTER 3

# EXPERIMENTAL INVESTIGATIONS INTO TCP PERFORMANCE

Chapter 2 dealt with the power control problem which is unique to the wireless world, and there being no existing protocols to handle the problem in ad hoc networks, it was necessary for us to examine and layout the fundamental design principles vis-a-vis power control. This chapter deals with the transport layer and TCP, which contrary to power control, has a large amount of research behind it, though not much experimental research on wireless ad hoc networks. Thus, we take a different approach in investigating TCP for ad hoc wireless networks. TCP performance is studied in a top-down investigative fashion through extensive experiments on a real testbed. The measurements are subjected to statistical testing. The resulting recommendations are expected to be directly useful in operating ad hoc networks.

## 3.1 Introduction

TCP can be given substantial credit for the success of the Internet. However, certain assumptions in the design of TCP caused problems when it was used over wireless links. There have been several studies of TCP performance over last hop wireless networks, and many reasonable solutions to fix TCP performance have been obtained. This work concerns itself with TCP performance over wireless multihop ad hoc networks. The simulators available for wireless networks are inadequate for a detailed study of wireless networks on several fronts, and exact analytical modeling of TCP over multihop wireless networks is also quite difficult.

This work, therefore, undertakes a thorough experimental investigation of TCP in a testbed of laptops with wireless cards.

This chapter is organized in the following manner. Section 3.2 presents a brief survey of the literature, and in the process motivates the need for an experimental study such as this one. Section 3.3 describes in detail the process of setting up experiments, data collection, management and analysis. We believe there are useful experiences to share in each of these arenas, as this is a very thorough experimental study which is not quite the norm in ad hoc networks. Section 3.4 describes the systematic process of investigating the effect of various parameters and conditions, one at a time, on TCP performance. This entails a description and discussion of various experiments performed under different conditions. This process leads us to identify the most critical recommendations which could impact TCP performance in ad hoc networks. These recommendations are then tested thoroughly and extensively over nine different experimental topologies of up to six hops, with several runs of each experiment. These are described in Section 3.5. The experimental observations are subjected to statistical tests to determine their significance, as is desirable for any scientific measurements. Section 3.5 describes the results of these tests in detail. The major contributions of this experimental study can be summarized as follows:

1. We conduct a thorough experimental investigation of TCP. In contrast, most previous studies have been almost exclusively based on simulations.

2. This work includes a thorough statistical analysis of the experimental measurements. This has not been the tradition at all in research related to ad hoc networks, even though it is common practice in most experimental studies in other fields.

3. We present concrete modifications to TCP over IEEE 802.11 wireless ad hoc networks. These are demonstrated to improve performance over the default TCP-Reno, through experimental and statistical testing. The anxious reader can jump to Section 3.5 to see the recommendations and results.

4. The output of this work also includes a set of tools to automate the complete process of experimentation, including set up, data collection, processing, archiving, and presentation of data.

## 3.2 TCP Over Wireless: A Brief Survey

This section gives an overview of the issues with TCP over wireless networks. There has been much work on TCP performance over "last-hop" or infrastructure wireless networks, where only the last hop is wireless. These networks first came into use when the existing cellular infrastructure was also used for carrying packetized traffic. These cellular networks are used predominantly for voice traffic and use a proprietary data link layer. Packet loss rates on wireless links are much higher than the corresponding wired links. The TCP protocol reacts to these losses in the same fashion as it would react to packet losses due to congestion, because it has been designed to react to losses in only that way. This observation, that TCP does not distinguish between congestion and loss, triggered a lot of studies and proposed modifications, many of which involved cross-layer interactions. For a classification and comparison of these approaches, see [53] and [54]. However, link losses due to high bit error rates can be shielded (to a large extent, if not completely) from the higher layers by the use of link layer ACKs. This prevents loss over a single hop from adversely effecting the whole end-to-end connection. The necessity of link-layer ACKs for an efficient wireless MAC protocol has been established through several studies, for example [2]. Thus, we assume that the underlying MAC protocol uses link-layer ACKs, and hence that losses due to link errors are very rare. The popular IEEE 802.11 standard does use link layer ACKS. In fact, many of the issues regarding TCP performance raised above were resolved or made irrelevant by the popularity of the IEEE 802.11 standard, and its use in last-hop wireless networks designed exclusively for packetized traffic such as the wireless LANs. Cellular data networks, also known as 3G, are still not a commercial success, probably underlining the hypothesis that carrying packetized data over networks designed for voice is extremely inefficient. Flarion Inc. is designing an all IP cellular data network, and their studies also suggest that TCP modifications suggested by previous studies may be unnecessary for such networks.

In multihop ad hoc networks, IEEE 802.11b is the dominant MAC protocol in use; hence, TCP not being able to distinguish between losses due to congestion and those due to channel errors is not of much consequence. However, mobility causes some new problems which have been studied in [55–57]. Node mobility can cause link breakage which may result in route failure. The network layer needs time for route re-computation which is typically larger than

the RTT. Meanwhile, TCP times out multiple times and also increases its timeout parameter exponentially. Hence, it does not probe for quite some time even after the route is restored, thus adversely affecting the utilization. The correct behavior seems to be to freeze all the timers when a route failure occurs, and resume sending when the route is restored. But how is TCP to figure out the events of route failure and route restoration? These are network layer events and a reasonable scheme is to require the network layer to send messages from the point of failure to the TCP source regarding such events. In fact, such mechanisms already exist in the current implementation of the IP stack in the form of ICMP messages. ATCP [14] uses this "smart network layer" approach to design a thin layer between TCP and IP which controls the TCP state machine. TCP-F [16] and TCP-BuS [58] are two other schemes which rely on explicit network layer signaling to suggest certain TCP modifications to improve performance during route failures. However, there exists opinion against TCP taking explicit support from the network layer, either because of infeasibility or architectural reasons. Such works advocate reliance on end-to-end statistics only. ADTCP [15] suggests a variety of such metrics, which, it is claimed, can be jointly used to figure out the cause of packet loss. TCP-DOOR [17] claims that out-of-order delivery is most likely due to route changes and not congestion, and uses precise end-to-end mechanisms for detecting out-of-order events.

However it is not clear if current TCP would perform well even over relatively static multihop wireless ad hoc networks. We seek to address this issue. There is very little work which attempts to address problems specific to TCP over IEEE 802.11b multihop wireless ad hoc networks, and even fewer experimental studies on a testbed. Fu et al. [59] contend that there exists an optimal TCP congestion window and conclude from their primarily simulation based studies that TCP window size actually grows much larger than the optimum, resulting in reduced spatial reuse. They suggest two techniques, link RED and adaptive pacing, to improve TCP throughput. Similar observations about excessive TCP congestion window growth have been made in [60]. They suggest the intuitive solution of sender congestion window clamping, which we have also arrived at independently. We evaluate our recommendation on a testbed, as opposed to the simulation based studies in [60]. Some results from experiments on an 802.11b ad hoc network have been presented in [61, 62].

## 3.3 Experimental Setup

The nodes in our testbed are reasonably powerful off-the-shelf laptops. They include Compaq Presario 2800T, Compaq Presario 1800T and HP Pavillion zv5240. The laptops have Redhat Linux 9.0 installed. For the experiments, we use the 2.4.19 Linux kernel which has been modified to log TCP parameters during the experiment. The wireless cards we use are Cisco Aironet 350 series PCMCIA cards [30]. These cards are preferred for two reasons. Firstly, they have six different transmit power levels of 1, 5, 20, 30, 50, and 100 mW, respectively, which is a great convenience in forming desired topologies. Secondly, the open source Linux driver for these cards provides access to various registers on the card which store statistics about various events corresponding to the IEEE 802.11 MAC protocol, thus making a more in-depth investigation possible.

We describe below some of the main problems encountered during setting up the experiments and the way we handled them. More problems and solutions are documented in the experiment blog maintained on the experiment homepage [37].

### 3.3.1 Scaling topologies with copper tape

Forming desired topologies is inconvenient since the communication range for these cards is too large. Even at the lower transmit power level of 1mW, the transmit range is up to 100 ft in an indoor environment. Thus, access is needed to multiple floors in a building to form multihop topologies. To reduce the transmit ranges we wrapped the Cisco cards with copper tape.[1] The copper tape we used was the "3M 1181 EMI Copper Foil Shielding Tape." When put on the Cisco wireless cards as shown in Figure 3.1, it changes the antenna impedance and causes an impedance mismatch between the card and the antenna, resulting in less power to be delivered on the air. Copper taped cards with multiple power levels gave us sufficient flexibility in creating desired topologies for our experiments in a hall or a couple of adjacent rooms.

[1] We are gratetful to Professor Jennifer Bernhard for suggesting this idea.

**Figure 3.1** A Cisco 350 card covered with copper tape.

**Figure 3.2** The inside of a Cisco Aironet 350 card.

### 3.3.2 Nonuniformity in card specifications

Commercial off-the-shelf wireless cards do not confirm tightly to their specifications. The inside of a typical Cisco Aironet 350 is shown in Figure 3.2. The patch antenna can be seen in the bottom right corner. Each card is actually a little different. The receive sensitivities vary and the communication range can be different for different cards. Thus, changing the cards could alter the topology significantly. Thus, one has to associate a card with a node throughout an investigation. One should also keep in mind that sometimes the performance of the cards may degrade after excessive use due to heating.

### 3.3.3 Interference from extraneous IEEE 802.11b sources

Our experiments were conducted in the Coordinated Science Laboratory building at the University of Illinois at Urbana. The building has an IEEE 802.11b Wireless network with about four access points on each of the four floors. This traffic could possibly interfere with our experiments, but it was not possible to shut off these access points as it is a production wireless network with many users. We investigated the possibility of using IEEE 802.11a, which would be immune to this interference as it operates in the higher 5 GHz frequency band. Its shorter communication range would also be an advantage for us as it would make forming multihop topologies more convenient. However at the time of the experiments there was no Linux driver for IEEE 802.11a cards which could work correctly in the ad hoc mode.

So we decided to go with the 802.11b Cisco Aironet 350 cards. We justified acceptance of the interference as it is likely to be present in a real environment too. However we carefully record all the interference during the duration of an experiment using wireless sniffers. Any experiment during which the level of extraneous interference was excessive is not analyzed along with other data. We use an open source wireless sniffer software called Kismet [63]. Two laptops are dedicated as sniffer nodes. The first one is sniffing on channel 10, which is the channel on which all the nodes of the testbed transmit and receive data. However, the eleven channels in 802.11b in North America are not all nonoverlapping as shown in Figure 3.3, taken from [3]. So we use another sniffer that hops over all the eleven 802.11b channels at a channel velocity of five channels per second. This sniffer misses some of the data packets on all the channels, but is able to detect excessive interference in the neighboring channels of our channel of interest, which is channel 10.

#### 3.3.3.1 Kismet versus Tcpdump

A natural question is why we do not we use a conventional wired network sniffing tool like tcpdump which can trace all packets when the card is put in the promiscuous mode. Promiscuous mode turns off the filtering mechanism in the network card, causing it to pass all packets to the operating system. But with wireless cards, even when the card is put in promiscuous mode, the driver gets packets only from the ESSID/network the card is currently

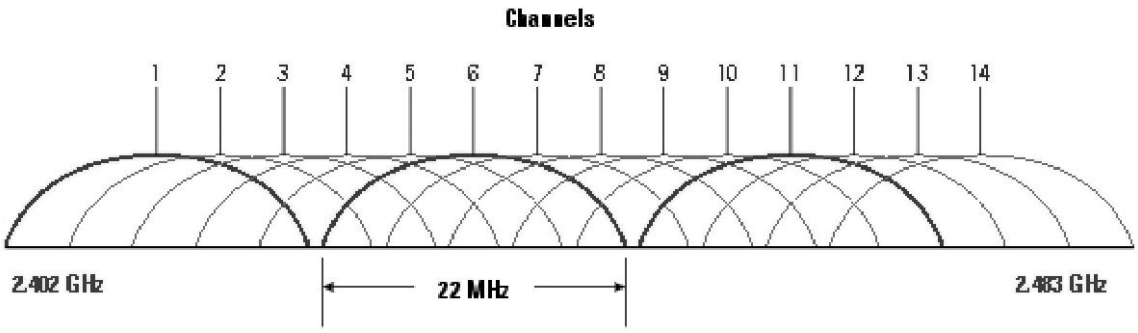



**Figure 3.3** IEEE 802.11b channel structure in North America.

associated with, and the card only passes the packets as 802.3 headers. This means tcpdump gets no 802.11 headers, no 802.11 management frames, and nothing from networks other than the one it is currently associated with. Rfmon is a special mode that reports all packets the wireless card sees, including management packets and packets from any network the radio can decode. Wireless sniffers like Kismet use the Rfmon mode to gather information about all the networks which it can sense, rather than just the network the card is associated with.

Another point to keep in mind is that tcpdump support for promiscuous mode for wireless cards is not stable. We have noticed that the Cisco cards simply stop sending data after a while when tcpdump is running in promiscuous mode on the same interface. If tcpdump is run without putting the cards in promiscuous mode, then things work fine. So we use tcpdump to collect data which a node is sending, receiving or forwarding, and we use Kismet to sniff all the traffic ongoing in the medium.

#### 3.3.3.2 Channel cannot be changed in ad hoc mode for the Cisco 350 series cards

The nodes of the testbed are on channel 10 (2.457 GHz) because we discovered that it is not possible to change the channel in ad hoc mode for the Cisco 350 cards. The cards were running version 4.25.30 of the firmware, which is the latest version available for Linux as of this writing. The card does not give any errors when an attempt is made to change the channel; in fact it appears to have changed the channel. But wireless sniffers reveal that in ad hoc mode these cards always transmit data on channel 10. In fact two nodes in ad hoc mode can always

communicate no matter what channel the card claims to be on. This is because they are always on channel 10.

Cisco cards are not appropriate for sniffing. This is because the firmware implements internal channel hopping which cannot be controlled, at least not when the card is in the RF monitor mode. Thus it is not possible to sniff on a particular channel or to change the channel hopping velocity. For the sniffing purposes we thus use the Lucent Orinoco Silver cards.

### 3.3.4 Traffic generation

For generating traffic we use the Nttcp [64] tool. Every node runs the Nttcp server on a pre-determined port which accepts TCP/UDP connections from Nttcp clients. Statistics about the connection are automatically collected at the source node at the end of the experiment.

### 3.3.5 Data collection

We have been very exhaustive in collecting data at different layers. At the MAC layer, the Cisco cards maintain 102 counters about different MAC events. The Linux driver can read all these registers and dump the information to the /proc file system. We sample this information once every second. The precise meanings of all these counters are not well documented. We have documented whatever we could decipher. It can be found in the "Analysis and Notes" section on the experiment home page [37].

At the network layer we collect various statistics about the transmit queue. These include the number of packets in the queue, the number of packets dropped, packet flow rate, etc. Linux has an extensive set of traffic control (tc) tools [65], which allow one to select one of many queuing disciplines and sample various statistics about them. The default discipline is first in first out (FIFO), but it is not a part of traffic control tool implementation, so statistics about this default queue are not easy to collect without modifying the kernel source. However if one uses the FIFO discipline that is part of the tc implementation, and is functionally similar to the default discipline, then statistics can be easily collected [66]. The default queue can be replaced and deleted respectively using the following commands:

```
tc qdisc add dev eth0 root pfifo limit 100
tc qdisc del dev eth0 root
```

The statistics about the qdisc can be collected using the following:

`tc -s -s -d qdisc show dev eth0`.

Tcpdump is run at each of the source nodes to collect tcp connection level information. Tcptrace [67] is a convenient tool to extract information from the tcpdump traces. As explained in Section 3.3.3, tcpdump is run in nonpromiscuous mode. Two Kismet wireless sniffers are run with the card in Rfmon mode to collect more traces about all ongoing connections.

No sniffing tool gives information about internal tcp parameters like congestion window (cwnd), slow start threshold (ssthresh), RTO, backoff etc. We modified the Linux kernel to dump these parameters for every connection into the /proc file-system whenever any of these values change. This enables us to accurately track these TCP parameters.

Besides these, we also record the output of netstat command, iwconfig command, iwstats command and the contents of /proc/net/snmp, /proc/net/dev, /proc/net/udp, /proc/net/tcp etc., every second for possible future use. Various other configuration settings of the network are recorded at the experiment start up as general logs. These include various settings in /proc/sys/net and other routing and interface information.

### 3.3.6 Data presentation and analysis

All the above information has been carefully archived for each experiment and is publicly available for download [37]. A CGI front-end is also available to process some of the data and generate comparison plots on the fly. Each experiment has a unique experiment number which we mention at various points in this paper, so that complete details about that experiment can be accessed online if desired.

### 3.3.7 Resources

We would like refer to certain useful resources that we discovered during the course of our work. The TCP implementation in the Linux 2.4 kernel goes beyond the standard textbook description and even beyond the latest RFCs. The Linux TCP implementation supports SACK,

TCP timestamps, Explicit Congestion Notification, Quick Acknowledgments, Rate-Halving, and algorithms for correcting incorrect congestion window adjustments. An excellent account of Linux specific TCP congestion control features is presented in [68]. While experimenting, it is immensely useful to go beyond TCP and understand the general structure of the networking code in the Linux kernel. An overview of the networking code is provided in [66]. Another useful document is [69] which explains the various networking sysctls in Linux.

There are various choices of packet sniffers available. Tcpdump and Ethereal are among the most popular traditional wireline sniffers. Wireless sniffers can extract far greater information by putting the card in Rfmon (RF monitor) mode. Kismet, Netstumbler, and BSD-Airtools are among the popular ones, though there are a whole array of tools available [70], thanks to the advent of Wardriving, which is the glorified name for the art of driving around looking for wireless networks. Many of these sniffers output data in a form which is backward compatible with the traditional sniffers, and hence can be analyzed by excellent tools like Tcptrace. For generating traffic, nttcp [64] is a slick little program. IPerf [71] is another choice which is well supported across various platforms.

## 3.4 The Process of Discovery

We started our experiments without any presumptions about TCP performance. Our first goal was to understand our experimental setup, and the effects of various parameters in simple topologies. Only then did we move on to more complex scenarios. There were lots of parameters that could be varied. As we proceeded with the experiments and understood the effects of some parameters reasonably clearly, we fixed their values so we could narrow down to the parameters which made an interesting difference to TCP performance.

We describe below these experiments and the choices made along the way as we discovered the more interesting issues with TCP performance.

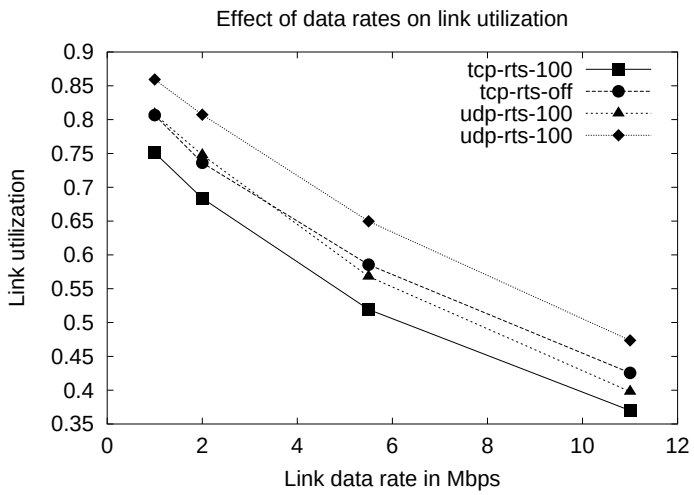



**Figure 3.4** Effect of IEEE 802.11b link data rates on channel utilization.

### 3.4.1 IEEE 802.11 data rates and the effectiveness of the four-way handshake

IEEE 802.11b supports data rates of 1, 2, 5.5, and 11 Mb/s, which are made possible by changing the modulation scheme. But the basic access rate which is used for the PLCP header and the RTS-CTS packets (since they are intended to be correctly received by all nodes in the vicinity) is 1 Mb/s. All the interframe spacings are also designed for 1 Mb/s. Thus, there is a standard enforced penalty for using higher link data rates, since only the DATA packets are sent at the higher rate while the control packets and the header are still sent at the lower basic data rate.

Figure 3.4 illustrates the link utilizations as observed in our experiments. The curve corresponding to "tcp-rts-100" indicates TCP traffic with the RTS threshold set to 100 bytes; i.e., RTS-CTS handshake is performed only for packets of size greater than 100 bytes. Note that ACKS are sent even when RTS-CTS handshake is turned off. At any given data rate, turning on RTS results in about a 5% decrease in link utilization. However, the link utilization itself is much lower at the higher data rate of 11 Mb/s compared to that at 1 Mb/s. The overhead of control packets appears exaggerated at 11 Mb/s. Also note that TCP has slightly lower utilization compared to UDP because of the end-to-end acks which do not count towards the goodput.

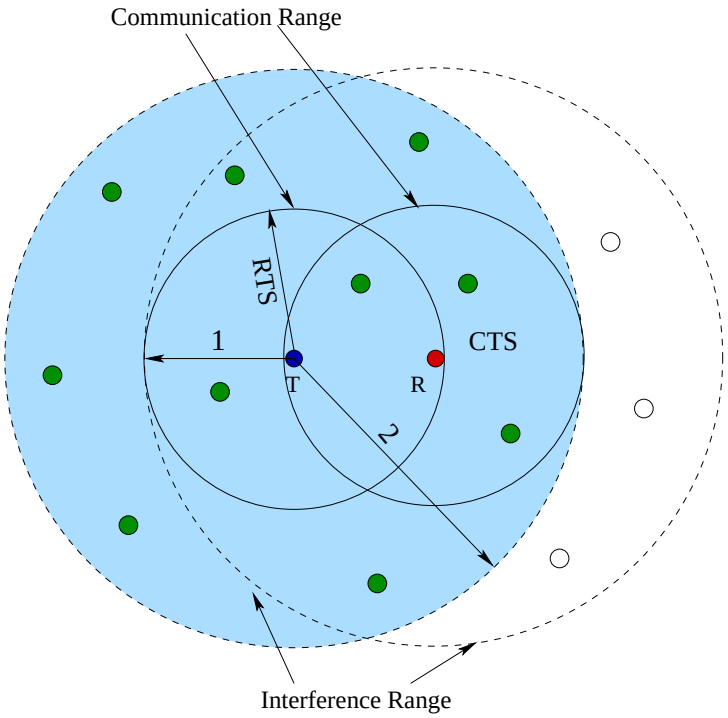



**Figure 3.5** RTS-CTS may be redundant if the carrier sensing range is at least twice the communication range.

In the above experiments we see that there is about a 5% degradation in goodput upon turning on the RTS-CTS handshake, at any given data-rate. However, with just two nodes there is absolutely no benefit in using RTS-CTS. Hidden terminal problems arise only in bigger topologies, where RTS-CTS could potentially be useful.

However, if the carrier-sensing range (the distance within which the carrier can be sensed) is at least twice the communication range (the distance within which the packet can be successfully decoded) and carrier sensing is on, then there is no benefit of using RTS-CTS. This is because, as illustrated in Figure 3.5, all nodes which could possibly be silenced by RTS or CTS packets are anyway silenced by the carrier of the DATA packet.

The observation in Figure 3.5, however, holds only when the DATA packets are sent at the same data rate as the RTS and CTS packets. At higher data rates, both the communication and the carrier sensing range are reduced. Add to all these complex interactions the fact that in a complex indoor environment with abundant multipath fading, carrier sensing range is very difficult to determine. Often there are regions of destructive interference where received

signal is very weak, and moving the nodes further apart actually improves the communication link. One also has to keep in mind that each card's circuitry is a little different and two cards could have different receiver sensitivities (see Section 3.3.2), resulting in different carrier sensing ranges. The popular assumption in many simulations, of the carrier sensing range being twice the communication range, is grossly inappropriate in such a complex environment.

Given these multiple factors, the reasonable way to determine the effectiveness of RTS-CTS handshake is by running multiple experiments over different topologies. The results are presented in Section 3.5.2. They indicate that turning off RTS-CTS increases throughput in all the topologies we tested.

### 3.4.2 Congestion window growth

One of the important features of TCP is its "self-clocking" property. According to this, a TCP sender adjusts its rate to fully utilize a nonshared bottleneck link by sending a new packet upon receiving a new ACK. Because of the self-clocking property, the congestion window opens up just enough to fill up the "network pipe," with the queues at the nonbottleneck links being empty most of the time. If all the links have the same capacity then there is minimal queuing in the network buffers in equilibrium.

A few assumptions underlying the above analysis however break down in multihop wireless networks. First, the capacity of a link in multihop wireless network depends on the fraction of time it can access the channel, which in turn depends on how busy the neighboring nodes are. Thus, the capacity of each link is time-varying; hence, there is no fixed bottleneck. Thus, TCP is trying to adjust its sending rate to the bottleneck capacity through an end-to-end mechanism, but the bottleneck itself could be changing, possibly at a time scale faster than that at which TCP can react. The result of this variation is an increase in the number of packets queued in the network buffers.

Second, the links in a wireless ad hoc network are themselves self-clocking if the MAC protocol used requires every packet to be acknowledged at every hop, as IEEE 802.11b does. At any link there is only one outstanding packet on the air, since the next packet cannot be sent until the acknowledgment for the current packet is received or the retry limit is exceeded.

This means each transmitter can have at most one-outstanding packet at any given time. For an $n$-hop connection, at most $n/2$ nodes can be transmitting at any given time due to the half-duplex nature of the nodes. The number of transmitters could be fewer due to hidden terminal effects, and due to the carrier sensing range being greater than the communication range. Thus, the capacity of the network pipe, excluding the router buffers, is at most $n/2$ packets per connection, where $n$ is the number of hops from the source to the destination for that connection. Thus the TCP congestion window does not need to be much larger than $n/2$ packets for any connection.

However our experiments reveal that the congestion window actually grows to a much larger value than $n/2$, depending on the transmit queue sizes and the MAC retry limits at various nodes. An excessive congestion window implies more outstanding packets in the network buffers, which leads to an overloading of the network. This results in an increase in the packet delays and delay jitters for all connections, and starvation of flows with fewer packets (the so-called mice).

#### 3.4.2.1 Analytical modeling

To get a better idea of the effect of an excessive congestion window growth on the end-to-end TCP throughput in steady state, we present an analytical model. This presentation is adapted from the analysis of closed migration process in [72], and is presented here for completeness.

Let us first consider the simple case of $N$ nodes forming a linear chain as shown in Figure 3.6. Suppose that a huge file is being sent from the source at one end to the destination at the other end. A sliding window protocol like TCP is used, but with the window size fixed at $W$ packets. Let us assume that the service times for the packets are independent and exponentially distributed with rate $\mu$ packets/s. Let us further assume that the MAC retry limit is high enough so that packet drops due to channel error are extremely rare.

In steady state, a congestion window of $W$ implies that that there are $W$ unacknowledged packets in the network which are queued in the buffers at the various nodes. The identity of the packets themselves is not important, since we are concerned only with the number of packets buffered at each node. Hence, the chain of Figure 3.6 can be considered as a closed system

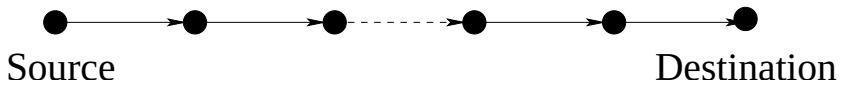



**Figure 3.6** A linear chain of nodes.

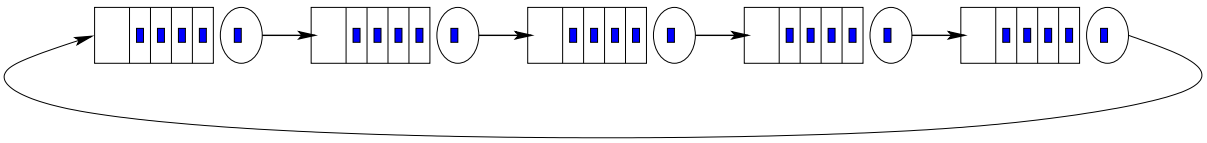


**Figure 3.7** The node chain modeled as a cyclic queue closed migration process.

since there are always $W$ outstanding packets in the system in steady state. Whenever a packet reaches the destination, the source injects a new packet which is equivalent to the destination handing the packet back to the source. This is shown in Figure 3.7.

Let $w_j$ be the number of packets queued at node $j$, and let $\mathbf{W}$ be the vector with components $w_1$,...,$w_N$. Note that each $w_j \geq 0$ and $\sum_{j=1}^{N} w_j = W$. Let $T_{jk}$ be an operator which, when acting on $\mathbf{W}$, moves a packet from node j to node k. That is,

$$T_{jk}(w_1, \ldots, w_j, \ldots, w_k, \ldots, w_N) := (w_1, \ldots, w_j - 1, w_j \ldots, w_k - 1, w_k + 1, \ldots, w_N). \quad (3.1)$$

For the system under consideration the only possible movement of a packet is from one node to the next in the chain in a cyclic fashion. Under this condition the random process $\mathbf{W}$ is a Markov chain with transition rates given by:

$$q(\mathbf{W}, T_{jk}\mathbf{W}) = \lambda_{jk}\phi_j(w_j), \text{ where} \quad (3.2)$$

$$\begin{aligned} \phi_j(w_j) &= 0, \ w_j = 0 \\ &= 1, \ w_j > 0 \\ \lambda_{jk} &= \mu, \text{ if } k = (j+1) mod\ N \\ &= 0 \ \text{ otherwise.} \end{aligned}$$

The process $\mathbf{W}$ is called a cyclic queue and belongs to the class of processes known as closed migration processes [72]. The steady state probability $\pi(\mathbf{W})$ can be verified to depend only on the total number of packets W and number of nodes N (it is of the form $\frac{c(W)}{N^W}$). Thus, for a given window size, all the states of a cyclic queue are equiprobable. This implies that $\pi(\mathbf{W})$

is just the reciprocal of the total number of states. The total number of states is equal to the number of ways of distributing W balls in N bins = $\binom{W+N-1}{W}$. Thus

$$\pi(\mathbf{W}) = \frac{1}{\binom{W+N-1}{W}}. \tag{3.3}$$

The throughput of the system (also known as the circulation velocity) is $\mu$ times the probability that a given node, say node $N$, is nonempty:

$$\nu_{circ} = \mu * P(w_N > 0) \tag{3.4}$$

$$= \mu \frac{\binom{W-1+N-1}{W-1}}{\binom{W+N-1}{W-1}} \tag{3.5}$$

$$= \frac{\mu W}{W+N-1}. \tag{3.6}$$

Figure 3.8 plots the throughput in packets/s, i.e., the circulation velocity against $W$, for a few different values of $N$. One observes that increasing $W$ gives diminishing returns in throughput. This behavior actually holds for more general queuing networks [73], even though a closed form expression is difficult to obtain. It is shown in [73], that the throughput for multi-station irreducible closed Markovian networks can be bounded by functions of the form $\frac{\alpha W}{W+v}$, where $W$ is the number of customers in the system. The coefficients $\alpha$ and $v$ can be obtained by solving two linear programs. The case for a linear chain, which can be solved exactly by direct combinatorial arguments (Equation (3.4)), corresponds to $\alpha = \mu$ and $v = N-1$. Note that the functional bound $\frac{\alpha W}{W+v}$ quickly tends to $\alpha$ as $W$ is increased. In fact, 90% of the asymptotic throughput is achieved at $W = 11v$. Thus, increasing the window size too much does not help. It may even hurt in practice as it increases the contention in the medium and also overloads the buffers in the network.

#### 3.4.2.2 Congestion window clamping

An obvious scheme to prevent congestion window overgrowth is to set a "clamp," i.e., an upper limit for the congestion window. The clamp should be different for every connection, and we suggest that heuristically that it be a small multiple of $n/2$, where $n$ is the number of hops for that connection. This is because, $n/2$ is the maximum possible number of packets outstanding in the network, excluding the packets in the router buffers. The next task is to

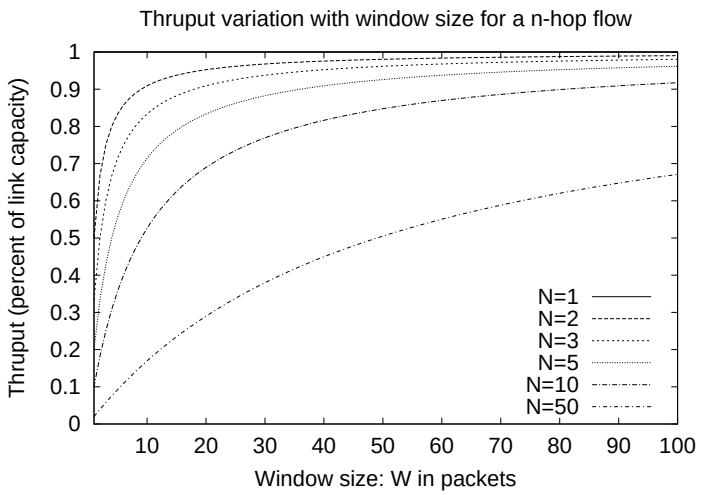



**Figure 3.8** Throughput for the cyclic queue process.

experimentally evaluate the modification we suggest. It turns out the Linux kernel already has the requisite mechanisms to implement congestion window clamping. The clamp can be specified for every destination in the routing table using the following command:

```
ip route replace 40.0.0.13/32 via 40.0.0.5 dev eth0 cwnd lock 2
```

The keyword "lock" is important. We manually set the clamp to different values and observe its effect on the TCP performance. The results in Figures 3.9 and 3.10 show a good match between the experimental results and the analytical predictions. The offset in the two curves represents the overhead of control packets which is not modeled in the analytical model. Throughput saturates very quickly as the congestion window clamp is increased, though the RTT keeps increasing because of queuing at the intermediate routers. Based on these measurements, a reasonable choice for the congestion window clamp seems to be $\lceil 3 * (n/2) \rceil$, where $n$ is the number of hops to be traversed by that flow.

#### 3.4.2.3 Window clamping and channel fading

The fact that a high congestion window could hurt was first reported in [59]. The idea of congestion window clamping has been recently suggested in [60]. They also suggest that the congestion window should never exceed the maximum bandwidth-delay product of the path (half the number of hops) since that is the maximum traffic carrying capacity of the path. We

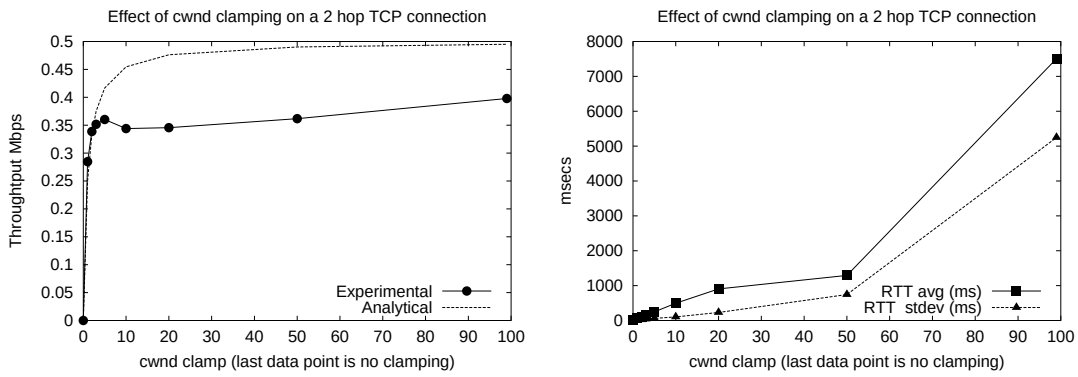



**Figure 3.9** Congestion window clamping for a two-hop TCP connection with 1 Mb/s links.

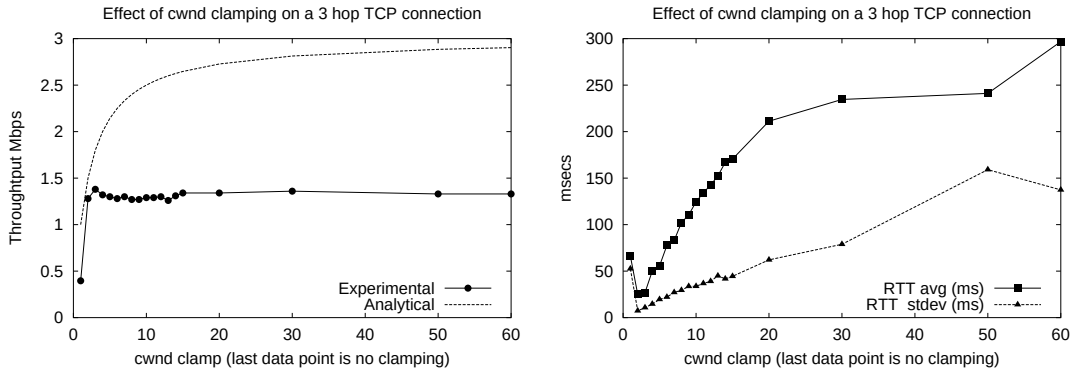



**Figure 3.10** Congestion window clamping for a three-hop TCP connection with 11 Mb/s links.

observe that this later assertion may not be true if one takes into account the variation in the channel due to fading and contention in the medium. We now present a counter-example where increasing the congestion window clamp beyond the so called "bandwidth delay product upperbound" increases TCP throughput. In fact, the TCP throughput increases linearly with the value of the congestion window clamp.

Consider a two-hop network of three nodes A, B, and C in a line. Let the channel fading be a deterministic on-off process with two states: good and bad. The state toggles every second. Suppose the channel state at receiver nodes B and C is completely out of phase, i.e., node B has a good channel when node C has a bad channel and vice versa. Let the maximum data-rate feasible on each link be 200 packets/s and assume that node B has a sufficiently large buffer. A TCP flow from node A to C through B with a window clamp of W packets would operate as follows on this channel. In the first second, W packets are transferred from node A to B, where

they are buffered since the channel at C is in a bad state. In the next second, the buffered packets are delivered to the destination node C. Thus, a throughput of W/2 packets/s can be achieved subject to the max possible throughput of 100 packets/s. However the hop based bandwidth delay product of the path is just 2 packets (equal to half the number of round trip hops), and clamping the window at that value would give a throughput of just 1 packets/s. Thus, the authors in [60] probably do not consider channels with fading when making their recommendations.

In general, the appropriate value for the clamp depends on the channel fading, the contention in the medium, and the number of TCP flows sharing the medium. We will show in Section 3.5 that the value of 3*(number of hops)/2 works better than the default unclamped case. But it may not achieve the best possible throughput.

### 3.4.3 Effect of packet sizes for the typical indoor wireless channel

Sending smaller packets incurs a greater packetization overhead, but bigger packets are more prone to channel errors causing a greater retransmission overhead [74]. To study the effect of packet sizes we first wanted to find out the largest packet that could be sent over the wireless link that would not be fragmented at any layer. For the Cisco 350 cards that magic figure turns out to be an MTU of 2276 bytes, and it took some detective work to discover it.

Ethernet cards allow an MTU of 1500 bytes, but wireless cards can send larger packets. The Linux driver for the Cisco cards we use allows setting the MTU to a maximum of 2400 bytes. But no packets can be sent if the MTU is set any larger then 2310 bytes. This includes the TCP/UDP and the IP headers. The maximum allowed fragmentation threshold for 802.11b is 2312 bytes, so one would expect that packets not larger then 2310 bytes would not be fragmented at the IP nor at the MAC layer. However, it turns out that the fragmentation threshold includes the size of the RTS packet (34 bytes) that would be sent before the data packet. Thus MTU should be set to 2310 - 34 = 2276 bytes for no fragmentation at any layer. Subtracting a further 20 bytes for the IP header and 8 bytes for the UDP header, this implies that the maximum size for UDP datagram at the application layer that will not suffer fragmentation is 2276 - 28 = 2248 bytes. The corresponding TCP MSS is 2224 bytes, as the TCP header is 32 bytes when the timestamp option is turned on.

We measured TCP and UDP throughput on a 1-hop connection for different MTUs. The data is displayed in Table 3.1. UDP datagram size was set to 2248 bytes and TCP automatically set its MSS to (MTU - 52) bytes. It appears that for the typical indoor channel, bigger packets are better as the error rates don't seem to be that high. The link data rate was 1 Mb/s. For the subsequent experiments we set the MTU to be 2276 bytes. Lowering the 802.11 fragmentation threshold also resulted in a degradation in performance. So we also set the fragmentation threshold to 2312 bytes which is the maximum allowed.

**Table 3.1** Effect of MTU on throughput.

| MTU (bytes) | TCP Throughput (Mb/s) | UDP Throughput (Mb/s) |
|---|---|---|
| 2276 | 0.83 | 0.88 |
| 1500 | 0.76 | 0.80 |
| 500 | 0.51 | 0.62 |

### 3.4.4 Unsuitability of minimum-hop routing for multihop wireless networks

As noted earlier in Section 3.4.1, it is difficult to characterize the "range" of a transmission in a wireless network. A significant fraction of the packets can be correctly received even at a relatively large distance, while there can be errors even at close distances. Further, these characteristics vary with time because of fading over the wireless channel. Thus, a weak link often exists between nodes which are far away, even though the throughput obtained on such links is low. A better end-to-end performance (in terms of throughput, delay, delay-jitter, energy, etc.) is often achievable by just avoiding the usage of such links. However, a minimum-hop routing protocol which functions by sending hello packets to neighbors will tend to select all the long range weak links to minimize the hop-count, hence resulting in poor performance. For example, in Figure 3.11 minimum hop routing will select the route A-C-E, even though the route A-B-C-D-E is likely to give a better throughput. The unsuitability of minimum hop routing protocols was also observed in the context of rate adaptive MAC protocols in [25].

This fact is easily observed by anybody attempting experiments on a real testbed. Nevertheless, minimum hop routing protocols are the most widely mentioned candidates for routing in ad hoc networks. The reason, we believe, is that they perform well in simulation studies, especially when signal strength, path loss, and rate as a function of range are improperly modeled

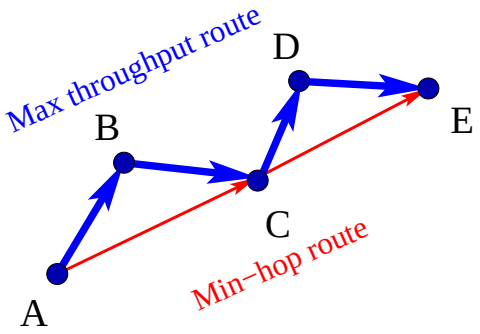



**Figure 3.11** Unsuitability of min-hop routing protocols for multihop wireless networks.

in the simulation model. This underlines the importance of experimental studies for wireless networks.

Minimum hop routing protocols have been studied in detail through experiments in [75]. The work also suggests an interesting metric: expected transmission count (ETX) instead of hop count for wireless networks.

A simple thresholding scheme to filter out the weak links may not work because of the so called relaying penalty. Most radios are half-duplex, and also the MAC protocol often requires a few neighboring nodes to be silenced. Because of these, just forwarding may incur a penalty of 1/2 to 1/4 (depending on the carrier sensing range) on the end-to-end throughput. Thus sometimes a long weak link may be preferable to two stronger but shorter links. The ETX metric naturally takes the relaying penalty into account. But if carrier sensing is turned on, then using short strong links imposes a high interference penalty since all the nodes which hear the carrier are silenced. Thus, choosing a good metric is a fairly complicated problem.

As far as this work is concerned, our focus is on TCP performance and its interaction with IEEE 802.11. To avoid being affected by the intricacies of routing, we fill in the routing tables manually to get reasonably strong links. The "strength" of a link is estimated by the loss rate and mean deviation of round trip times of ping packets sent at the rate of 100 packets a second. In our topologies we ensured less than 1% loss rate and a mean deviation of less than 1 ms, for each link.

### 3.4.5 Complications with automatic rate adaptation

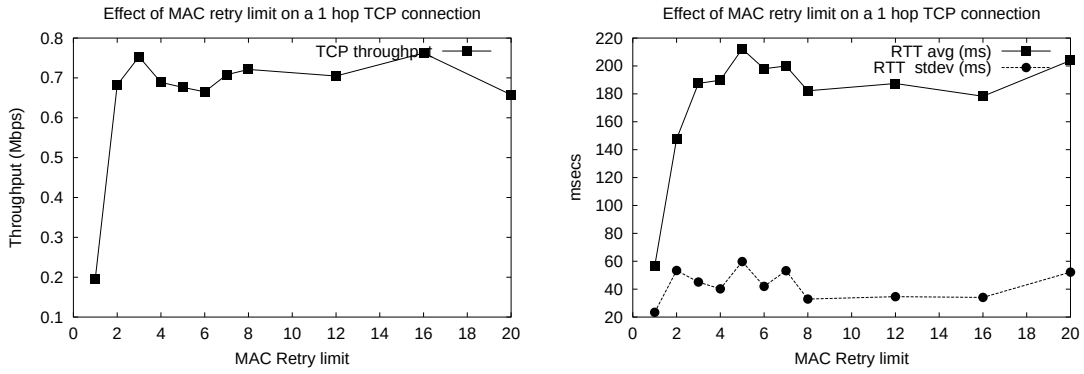
IEEE 802.11b has four available bit rates: 1, 2, 5.5, and 11 Mb/s. The rate can be fixed to one of these, or set to "auto" which turns on the auto rate fallback feature supported by most cards. The algorithm for auto rate selection is not part of the standard and is vendor specific. For example, [76] selects future rates based on the history of rates used. Such schemes typically give performance improvements in access points networks.

But matters are more complicated in ad hoc networks. If various links along a path are using different data rates, then simple routing metrics like minimum hop are rendered unsuitable, since they will choose the slow and unreliable links as noted in Section 3.4.4. The rates need to be explicitly taken into account in designing the metric (see, for example [77]). However, there are more problems in practical implementations. The range is different at each rate since the receiver threshold for detection is different, and all the MAC packets as well as broadcast packets are always sent at the base rate since they are intended for multiple receivers. Thus, even if higher rate links are selected by a smart scheme, the transmission and reception still silence a larger area corresponding to the range at the base rate.

The point is that turning on the simple auto-rate adaptation could actually degrade performance in ad hoc networks. It may even cause network partitions. Hence, in all our experiments we turn off auto-rate adaptation and set the link data rate to a fixed value, since our focus is to investigate TCP performance.

### 3.4.6 Effect of MAC retries

Typical bit error rates in the wireless channel ($> 10^{-5}$) are four to five orders of magnitude higher than wired Ethernet. Thus a hop-by-hop acknowledgment for every DATA packet is crucial so that an entire end-to-end connection would not suffer from a packet loss at a single hop. In fact, at every hop a packet is retried a few times until the acknowledgment is received. If acknowledgment is not received after *MACRetryLimit* number of retries then the packet is dropped. There are in fact two limits: *MACShortRetryLimit* for RTS packets, and *MACLongRetryLimit* for DATA packets.




(a) Effect on throughput.

(b) Effect on delay and delay jitter

**Figure 3.12** Effect of MAC retry limit on a 1-hop TCP connection.

We now study the effect of varying retry limit on TCP throughput. In our experiments the short and long retry limits were always equal to each other. Figure 3.12 shows the effect of changing retry limit on a 1-hop TCP connection. Here the first transmission of a packet is not counted as a retry. Thus, a retry limit of 1 implies that a packet is tried at most twice before being dropped. The minimum value of the retry limit that can be set on the Cisco cards is 1. Throughput is only 20% of link bandwidth when the retry limit is 1, but quickly rises as the retry limit is increased, as seen in Figure 3.12(a). Increasing the retry limit further yields diminishing returns for throughput as the loss probability decreases exponentially when the retry limit is increased. However, the delay and delay jitters increase significantly as the retry limit is increases, as seen in Figure 3.12(b).

It may appear that the highest possible setting of retry limit is good. If the channel is good then the packet goes through soon and the retry limit is not called into effect. When the channel is bad, the retries shield the higher layers from the errors at the link level. Infinite persistence is in fact good when the wireless link is used exclusively by two nodes [53]. But when the link is shared by a number of nodes, then it may be possible that the channel at some receiver is bad, but is good at another receiver. Such a scenario is illustrated in Figure 3.13. In such a case, where the link is shared, infinite persistence leads to the head of the line blockage and decreases utilization of the channel. The problem is further compounded by the fact that

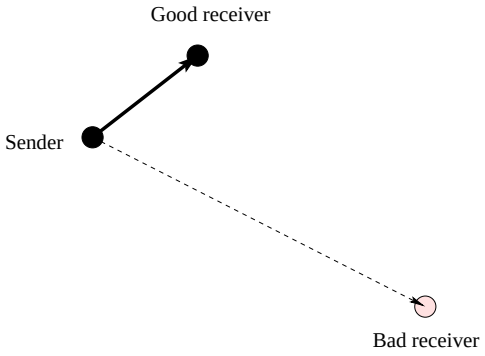



**Figure 3.13** An example topology when a sender is simultaneously sending packets to two receivers, one with a good channel state and another with a bad channel state.

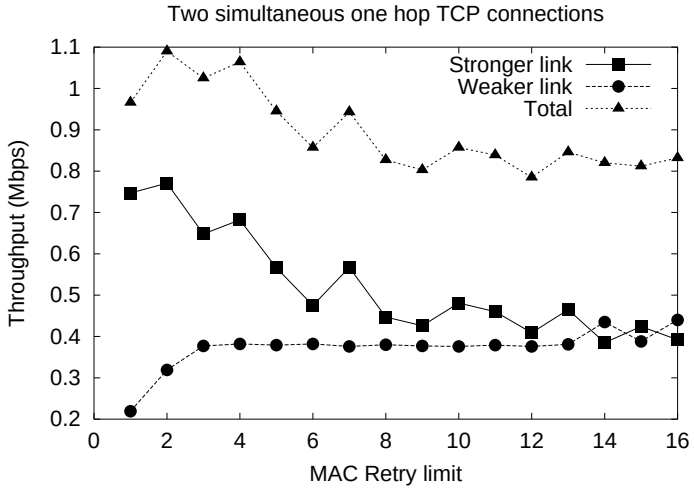



**Figure 3.14** Effect of MAC retry limit on two simultaneous TCP connections over the topology of Figure 3.13.

each retry is also accompanied by a doubling of the contention window at the MAC layer from which the random backoff is selected.

An experiment was done over the topology of Figure 3.13 in which two TCP flows were run simultaneously over the strong and weak links. The MAC retry limit was varied from 1 through 16 in subsequent runs. The TCP throughput is shown in Figure 3.14. As predicted, the TCP flow over the stronger link, and hence the total throughput, benefits if the packets over the weak link are not retried too often. It appears that a retry limit of four is good both with regard to the total throughput achieved and fairness. Thus, too high a retry limit is probably

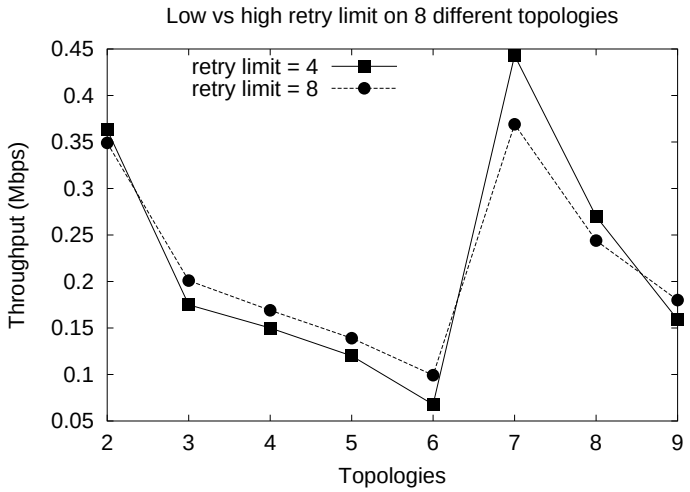



**Figure 3.15** Comparing throughputs for MAC retry limit of four and eight for topologies two through nine of Figure 3.17 (p. 76

not good in scenarios where a node could be transmitting to multiple neighbors whose channel qualities differ greatly.

However, the TCP connections in the experiment described above were over single hops. We did several more experiments over the topologies described in Section 3.5.2. The case of MAC retry limit equal to four was compared to the default case of MAC retry limit equal to 8. Figure 3.15 shows the throughput of the two schemes, averaged over 15 runs for the two cases. We observe that a low retry limit does better only for topologies with intersecting flows of at most three hops. A high retry limit gives better performance when there is a single flow, or when there are flows with many hops.

We perhaps need a mechanism to be able to set the retry limit to a different value for every receiver, with this value updated dynamically based on the channel state estimate at the receiver. This would constitute a cross-layer design modification for improving TCP throughput, and as discussed in Chapter 5 one needs to holistically examine its consequences. In any case, in the absence of any such mechanism, we recommend that a MAC retry limit of 8 be used for TCP connections.

Note that the above investigations assume a FIFO scheduling policy at the link layer. If more sophisticated scheduling at the MAC layer is feasible, e.g., scheduling based on channel quality at the receiver, then performance improvements are possible.

### 3.4.7 Effect of varying the router queue length

The basic purpose of a buffer in a router is to make an end-to-end flow oblivious to temporary nonavailability of a link, i.e., to shield it from short term link variations. One of the causes of the temporary nonavailability is the fact that the buffer or the queue is shared by many flows. Router buffers in the Internet need to be large since they are shared by a potentially large number of flows. In an ad hoc network, each node is a router, and there are not too many flows sharing a router buffer. The other causes of nonavailability in a wireless network are channel fading and contention by neighboring nodes as it is a shared medium. Thus, buffering in a wireless ad hoc network is mainly to deal with channel variations, while dealing with multiple flows sharing a link is only a secondary concern.

A buffer overflow is a signal to TCP to stop its congestion window growth, since TCP will keep pumping data into the network until at least one of the buffers is full. Large router queues mean that storage of TCP packets is transferred from the source node to the intermediate router nodes in the network. This prevents other flows from getting a fair use of the network. For example, real time flows will see a huge wait at each of the intermediate nodes and hence see long round-trip times. The effect of varying the size of the transmit queue, or the interface send buffer, on a two-hop TCP connection is shown in Figures 3.16. On increasing buffer sizes the throughput increases by less than 2% while the round trip delay and delay jitters increase by at least two orders of magnitude.

Reducing the buffer sizes so that it is just enough to account for the channel variation can actually improve the performance of a single TCP flow. But the small buffers will be mostly kept full by TCP, and consequently a nonbandwidth intensive flow sharing the same links will suffer. One approach is to drop packets when the queue size exceeds some virtual queue size, which is smaller than the real queue. Some active queue management schemes use this idea [78]. However, for ad hoc networks there are other ways of keeping the buffer occupancies small. Congestion window clamping, explored in Section 3.4.2.2, is one such mechanism.

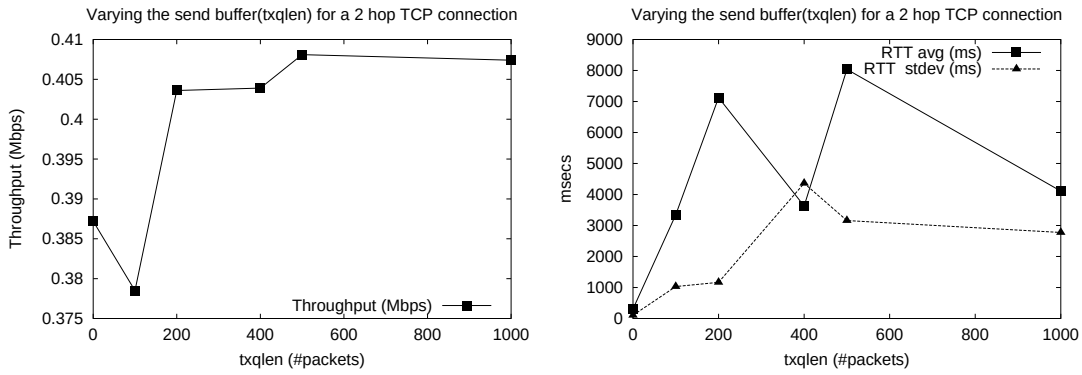



**Figure 3.16** Effect of varying the queue size on a two-hop TCP connection.

### 3.4.8 MAC does not distinguish between contention and loss

Before every retry, IEEE 802.11 MAC doubles its contention window, which is the time from which a random backoff is selected. This is important for the stable operation of the network. When the contention in the medium is too high, exponential backoff is essential for stability. However, IEEE 802.11 backoff may be too conservative. A transmitter backs off on the occurrence of one of these four events:

1. Physical carrier sensed to be busy.
2. Virtual carrier (NAV) sensed to be busy.
3. Sender timed out waiting for a CTS after sending an RTS.
4. Sender timed out waiting for an ACK after sending a DATA packet.

Of these four events, the first two are definite indications of contention in the medium, while the last one most likely indicates loss of a packet due to channel error if RTS-CTS has been turned on. This is because the RTS-CTS handshake is expected to reserve the medium before the DATA packet is sent, and so one would not expect any neighbors to contend for the channel until the transmission is complete. This is by and large true except for some boundary cases resulting mostly from the fact that the carrier sensing range is greater than the communication range. The third event, CTS timeout, could be caused either by channel error or by contention. To be conservative, the contention window could be doubled upon the occurrence of the third

event, but doubling it on an ACK timeout is unnecessarily conservative. The packet should be retried without doubling the contention window.

Thus we recommend that if RTS-CTS is turned on, the contention window should not be doubled when a DATA packet is retransmitted because of ACK timeout.

### 3.4.9 TCP selective acknowledgment (SACK) option

Selective acknowledgments are widely regarded as beneficial for TCP performance in wireline networks and last hop wireless networks. However in our scenarios it is not obvious that SACKs are always good. This is because the use of SACKs can promote TCP congestion window inflation by reducing timeouts due to reordering. Turning off SACK can result in decrease of congestion window, which could improve performance in the absence of any other mechanism to prevent the overloading of network buffers.

### 3.4.10 Other parameters

We have not studied the effect of several other parameters on TCP performance. We briefly mention them here. In all our experiments WEP was turned off, i.e., there was no encryption at the MAC layer. We also turned off all the power-saving mechanisms in the wireless cards. TCP send and receive buffers were set to 2 MB each, and the window scaling option was turned on.

## 3.5 Results and Recommendations

Based on the investigations so far, we identify the most critical recommendations for TCP over ad hoc networks, and subject them to extensive experimental and statistical evaluations. These recommendation are as follows:

1. A hard limit or clamp should be enforced on the TCP sender congestion window. We recommend clamping the congestion window to $\lceil \frac{3}{2}n \rceil$, where n is the number of hops to be traversed by that flow. This mechanism was investigated in Section 3.4.2.2.

2. The RTS-CTS mechanism should be turned off, at least when operating at highest data rate of 11 Mb/s for IEEE 802.11b. Section 3.4.1 investigated the effect of RTS-CTS in detail, leading to this recommendation.

3. The TCP selective acknowledgment option should be turned on. The effect of SACKs was discussed in Section 3.4.9.

We evaluate all possible combinations of these modifications on TCP performance in different scenarios or topologies. During these experiments several other parameters were fixed at values which were determined to be good from a smaller set of experiments described in Section 3.4. These additional recommendations are as follows:

1. MAC retry limit, i.e., the maximum number of retransmission attempts for a packet at the MAC layer is set to 8. A lower MAC retry is possibly useful only when there are multiple intersecting flows of at most three hops. Section 3.4.6 investigated this issue in detail.

2. The interface buffer at the nodes be set to 100 packets. Our recommendations are designed to keep the buffer occupancy low, and hence reduce end-to-end delays. See Section 3.4.7 for a discussion of various types of buffers and their effects on performance.

3. The MTU is set to 2276 bytes which is the maximum possible value for the Cisco 350 cards such that there is no fragmentation of packets at the network layer or the MAC layer. Section 3.4.3 explored this issue in detail.

4. The fragmentation threshold at the MAC layer is set to 2312 bytes, the maximum possible value allowed by the IEEE 802.11 specifications [3]. Packets of size greater than the fragmentation threshold are fragmented at he MAC layer and sent in smaller packets. A high value for this threshold can reduce the packetization overhead. This issue was discussed in Section 3.4.3.

### 3.5.1 Design of experiments

We now describe the results of our experimental evaluations of the proposed modifications. In the language of statistics, a *treatment* is a control administered to an experiment. Treatments

**Table 3.2** Description of treatments.

| Code | Name | Explanation |
|---|---|---|
| 0 | default | No clamping, sack off, RTS-CTS on |
| 1 | all | all=clamp+sack+rts_off |
| 2 | clamp | Sender cwnd clamped to 3*(number of hops/2) |
| 3 | rts_off | RTS-CTS mechanism switched off |
| 4 | sack | TCP selective acknowledgment turned on |
| 5 | clamp+rts_off | Treatments 2 and 3 combined |
| 6 | clamp+sack | Treatments 2 and 4 combined |
| 7 | sack+rts_off | Treatments 3 and 4 combined |

are represented by values of the independent variable (controlled by the experimenter) also called *factor*, which is typically a categorical variable. The eight treatments (in our case, protocol modifications) we have selected are all possible combinations of the three modifications: congestion window clamping, RTS-CTS handshake off, and TCP selective acknowledgment option on. They are listed in Table 3.2.

For each of the treatments we measure several *response variables* at different layers in the stack. In the subsequent analysis though, we focus on three main response variables: the end-to-end TCP throughput, the average delay or RTT and standard deviation of RTT (also called delay jitter).

The response variables are measured for each treatment on nine different experimental topologies, which are called *plots* in the language of statistics. We can also refer to them as scenarios or topologies. The first six plots or scenarios consist of a single TCP flow of one through six hops, and the next three plots consist of two intersecting flows of two, three, and four hops, respectively. These are shown in Figure 3.17.

For each plot, 15 runs of an experiment are performed for each treatment, and all the response variables are recorded, which totals to 1080($= 15 \times 8 \times 9$) experiments. Each experiment lasts about 100 s and consists of transferring a huge file over a TCP connection from each source to its destination. The size of the file is different for different plots. The experimental settings have been described in greater detail later in Section 3.3.

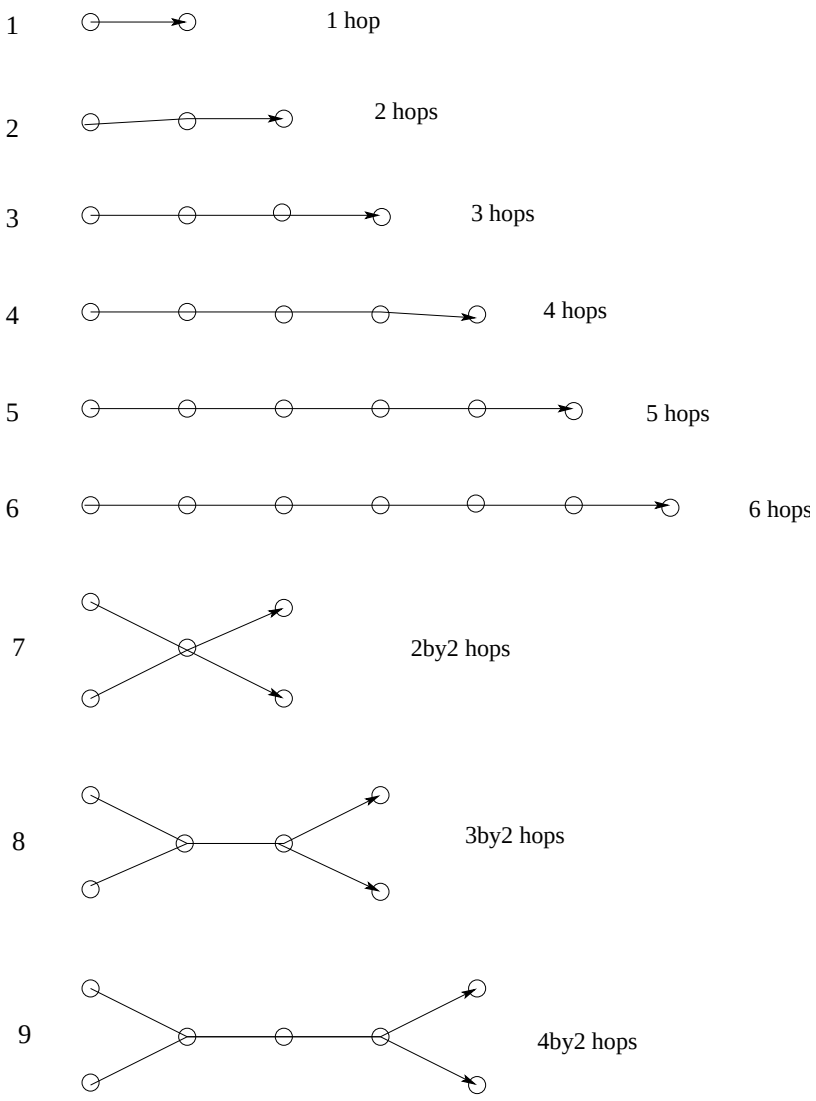



**Figure 3.17** The experimental topologies or plots.

### 3.5.2 Summary of measurements

For each plot, 15 runs/repetitions of an experiment are performed for each treatment. In calculating the means for throughput, average RTT (delay), and standard deviation of RTT (delay jitter) we discard the runs with the maximum and the minimum values of throughput. This prevents outlier readings from affecting the calculation of the means greatly. The means for all the three response variables are plotted in Figure 3.18 (p. 78), and displayed in Table 3.3. For the topologies with two TCP flows (2by2, 3by2 and 4by2), the mean throughput listed is

the total of the throughput for the two TCP flows, whereas the delay and delay jitter values listed are averaged over the two flows.

From a visual inspection of the graphs in Figure 3.18 and observing the values in Table 3.3, the following observations can be made. Treatments 1, 2, 5, and 6 seem to achieve considerably lower values for delay and delay jitter, while treatment 7 and 3 seem to get better throughput, though 1 and 5 are also close. This implies that congestion window clamping results in a significant reduction in delay and delay jitter, while turning RTS-CTS handshake off results in a small but noticeable increase in throughput. But these variations may result from random experimental effects. To draw reliable conclusions, we need to subject these experimental observations to statistical analysis.

**Table 3.3** Means for the response variables.

| Treatment | 1hop | 2hop | 3hop | 4hop | 5hop | 6hop | 2by2 | 3by2 | 4by2 |
|---|---|---|---|---|---|---|---|---|---|
| Mean (over runs) of throughput | | | | | | | | | |
| 0 | 4.12 | 2.03 | 1.28 | 1.02 | 0.78 | 0.53 | 1.91 | 1.78 | 1.08 |
| 1 | 3.33 | 2.06 | 1.45 | 1.12 | 0.86 | 0.58 | 1.94 | 1.60 | 1.12 |
| 2 | 3.20 | 1.92 | 1.36 | 1.02 | 0.79 | 0.52 | 1.78 | 1.48 | 1.06 |
| 3 | 4.54 | 2.19 | 1.41 | 1.08 | 0.84 | 0.53 | 2.05 | 1.84 | 1.23 |
| 4 | 4.29 | 2.08 | 1.34 | 1.02 | 0.78 | 0.53 | 1.97 | 1.72 | 1.10 |
| 5 | 3.33 | 2.06 | 1.45 | 1.11 | 0.90 | 0.55 | 1.90 | 1.59 | 1.14 |
| 6 | 3.18 | 1.92 | 1.32 | 1.03 | 0.80 | 0.52 | 1.75 | 1.47 | 1.03 |
| 7 | 4.64 | 2.26 | 1.45 | 1.11 | 0.89 | 0.57 | 2.11 | 1.94 | 1.21 |
| Mean (over runs) of average RTT | | | | | | | | | |
| 0 | 68.9 | 750.6 | 1237.1 | 838.6 | 180.9 | 304.1 | 743.0 | 569.4 | 350.4 |
| 1 | 8.6 | 17.1 | 49.2 | 88.6 | 133.8 | 183.4 | 43.4 | 100.7 | 154.5 |
| 2 | 9.1 | 18.5 | 53.3 | 97.1 | 143.1 | 195.3 | 48.8 | 109.3 | 161.6 |
| 3 | 39.3 | 687.8 | 1148.8 | 919.1 | 170.5 | 270.4 | 702.6 | 513.3 | 340.3 |
| 4 | 70.2 | 655.9 | 1074.5 | 863.1 | 186.0 | 333.4 | 790.3 | 519.5 | 377.0 |
| 5 | 8.6 | 17.1 | 49.4 | 89.9 | 132.9 | 182.6 | 45.1 | 100.4 | 152.5 |
| 6 | 9.1 | 18.4 | 55.0 | 96.8 | 145.5 | 194.6 | 50.3 | 110.5 | 169.3 |
| 7 | 40.3 | 590.3 | 1077.5 | 760.0 | 174.2 | 291.3 | 716.6 | 520.4 | 342.2 |
| Mean (over runs) of standard deviation of RTT | | | | | | | | | |
| 0 | 78.3 | 205.6 | 307.4 | 248.9 | 69.6 | 191.2 | 277.0 | 274.5 | 241.2 |
| 1 | 3.9 | 5.5 | 16.0 | 22.9 | 41.9 | 77.5 | 21.4 | 40.5 | 60.3 |
| 2 | 3.7 | 6.0 | 16.7 | 25.2 | 47.1 | 87.2 | 23.4 | 41.4 | 62.3 |
| 3 | 34.5 | 182.5 | 282.1 | 378.8 | 71.5 | 191.9 | 252.3 | 245.8 | 251.0 |
| 4 | 74.4 | 203.0 | 286.2 | 332.2 | 70.8 | 246.9 | 240.8 | 220.3 | 241.8 |
| 5 | 3.9 | 5.4 | 15.4 | 23.7 | 40.3 | 74.8 | 22.1 | 40.7 | 57.3 |
| 6 | 3.8 | 5.8 | 17.8 | 24.9 | 43.2 | 77.8 | 27.5 | 41.4 | 61.5 |
| 7 | 36.3 | 160.5 | 249.5 | 326.0 | 70.4 | 189.5 | 247.2 | 256.7 | 245.9 |

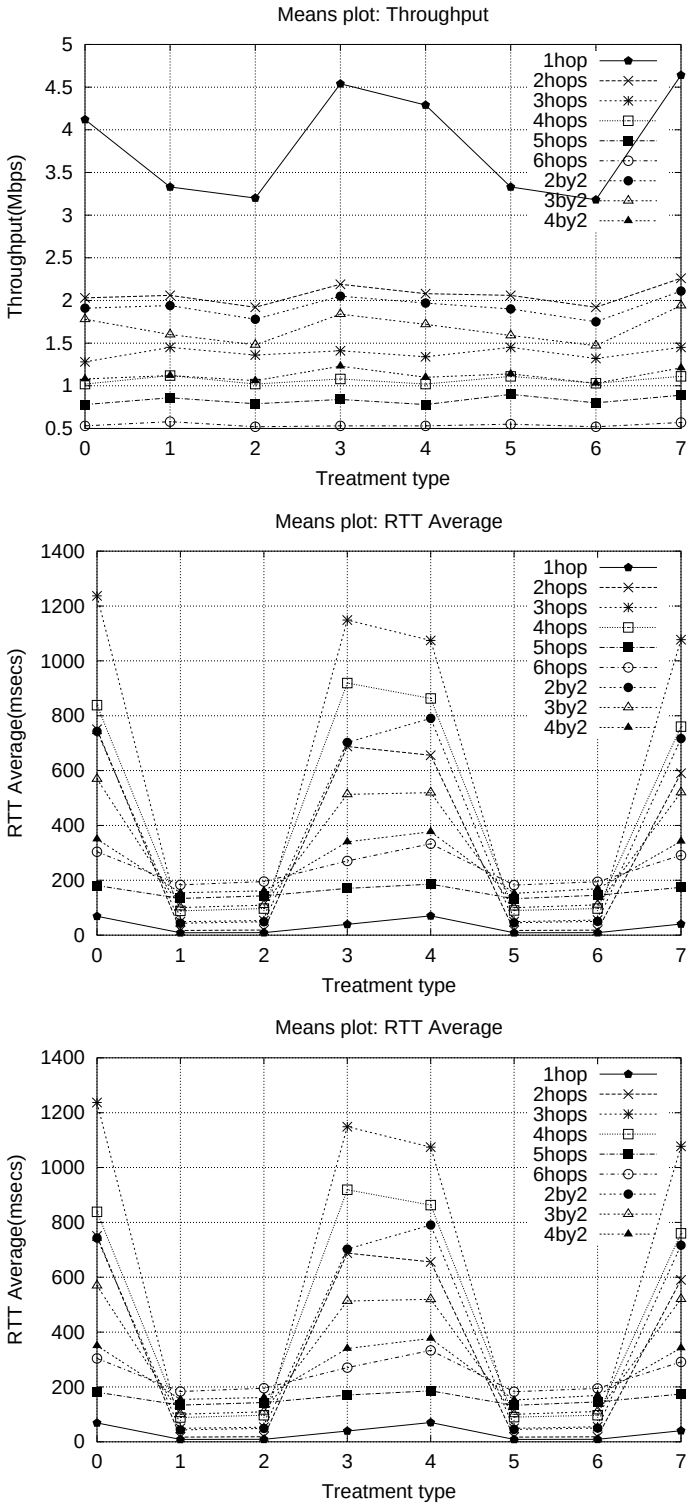



**Figure 3.18** Mean values (over runs) of throughput, average RTT, and standard deviation of RTT.

### 3.5.3 Statistical analysis

Before describing the results of the statistical analysis, some definitions [79] are in order.

- **Null hypothesis:** The null hypothesis for a statistical test is the assumption that the test uses for calculating the probability of observing a result at least as extreme as the one that occurs in the data at hand.
- **Significance level:** The significance level (also known as the $\alpha$-level) of a statistical test is the preselected probability of (incorrectly) rejecting the null hypothesis when it is in fact true. Usually a small value such as 0.05 is chosen.
- $p$ **value:** In a statistical hypothesis test, the $p$ value (sometimes called just significance) is the probability of observing a test statistic at least as extreme as the value actually observed, assuming that the null hypothesis is true. This probability is then compared to the preselected significance level of the test. If the $p$ value is smaller than the significance level, the null hypothesis is rejected, and the test result is termed significant.

We estimate the significance of the experimental results using ANOVA techniques [80–82]. ANOVA works by splitting the observed data values into "components" that are attributable to different levels of the factor or treatment. ANOVA tests the null hypothesis that there is no difference in the means for different levels of the factor. We use SPSS [83] for our computations. ANOVA makes the following assumptions:

1. **Independence:** Observations are assumed to be independent. In our experiments each observation is obtained from a new TCP connection at a different time. For a given treatment, the knowledge of any one observation does not give any information about others. Thus, independence of observations is a reasonable assumption for our data.
2. **Normality:** ANOVA assumes that the distributions of the response variables in each of the groups are normal. It is however robust to nonnormality if the distributions are symmetric or skewed in the same direction. A popular test for normality is the Shapiro-Wilk test [82], which examines the null hypothesis that the data is normally distributed. Thus, for ANOVA assumptions to be valid, we want the Shapiro-Wilk statistics to be

nonsignificant, i.e., have a $p$ value greater than 0.05. Table 3.4 displays the $p$ values from the Shapiro-Wilk tests for our data, and for most (about two-third of the cases) of the treatments and topologies they are indeed nonsignificant. In a few cases (about one-third) the data is not normal, but given the robustness of ANOVA to nonnormality, we conclude that it is an appropriate test for our data. In case of significant deviation from normality, a nonparametric test like the Kruskal-Wallis test can be used.

3. **Homogeneity of variance:** ANOVA assumes that the variances for groups that are being tested for equality of means is the same. This assumption can be tested using Levene's test [82]. The null hypothesis for the test is that the group variances are all homogeneous. Thus, for the assumptions made by ANOVA to be valid, we want the Levene statistics to be nonsignificant, i.e., greater than 0.05. The $p$ values from the Levene's test for our data are displayed in Table 3.5, and are actually significant for many of our response variables. Thus, modifications to that standard ANOVA $F$ test, like the Welch and the Brown-Forsythe tests, may be more appropriate for our data. However standard ANOVA is also robust to lack of homogeneity of variances if it is balanced; i.e., the sample sizes for the treatments are equal, which is the case for our data. We run all the three tests using SPSS.

Table 3.6 (p. 83) lists the $p$ values (statistical significance), computed using three different tests: the standard ANOVA $F$ test, Welch test and the Brown-Forsythe test. The Welch and the Brown-Forsythe tests do not make the assumption of the homogeneity of variances. We find that the null hypothesis is rejected by all the three tests for all the three measured variables for all the topologies, since the $p$ values are less than the $\alpha$-level of 0.05.

ANOVA tests the null hypothesis but gives no further information on which treatment is better. This information can be obtained by running multiple comparison post hoc tests. There are various tests available which are suitable under various circumstances. Pairwise comparison tests compare the differences between each pair of means, whereas range tests identify subsets of means that do not differ from each other. Another classification of post hoc tests divides them into single-step and stepwise tests. Stepwise tests do not provide a confidence interval for the pairwise comparisons but are more powerful; see [84] for further details.

**Table 3.4** $p$ values for Shapiro-Wilk test for normality.

| Treatment | 1hop | 2hop | 3hop | 4hop | 5hop | 6hop | 2by2 | 3by2 | 4by2 |
|---|---|---|---|---|---|---|---|---|---|
| Throughput | | | | | | | | | |
| 0 | 0.345 | 0.064 | 0.009 | 0.410 | 0.209 | 0.677 | 0.027 | 0.482 | 0.125 |
| 1 | 0.713 | 0.001 | 0.004 | 0.133 | 0.974 | 0.098 | 0.884 | 0.974 | 0.064 |
| 2 | 0.004 | 0.006 | 0.052 | 0.021 | 0.182 | 0.198 | 0.155 | 0.016 | 0.511 |
| 3 | 0.369 | 0.639 | 0.009 | 0.261 | 0.957 | 0.168 | 0.032 | 0.166 | 0.066 |
| 4 | 0.455 | 0.000 | 0.074 | 0.056 | 0.395 | 0.400 | 0.380 | 0.309 | 0.007 |
| 5 | 0.357 | 0.108 | 0.000 | 0.229 | 0.774 | 0.601 | 0.006 | 0.552 | 0.291 |
| 6 | 0.005 | 0.000 | 0.002 | 0.187 | 0.423 | 0.063 | 0.273 | 0.159 | 0.491 |
| 7 | 0.050 | 0.007 | 0.003 | 0.757 | 0.327 | 0.707 | 0.616 | 0.266 | 0.014 |
| Average RTT | | | | | | | | | |
| 0 | 0.157 | 0.593 | 0.008 | 0.008 | 0.903 | 0.379 | 0.419 | 0.626 | 0.165 |
| 1 | 0.116 | 0.004 | 0.017 | 0.276 | 0.270 | 0.943 | 0.896 | 0.094 | 0.045 |
| 2 | 0.003 | 0.002 | 0.011 | 0.020 | 0.039 | 0.388 | 0.171 | 0.193 | 0.559 |
| 3 | 0.163 | 0.070 | 0.030 | 0.466 | 0.065 | 0.106 | 0.964 | 0.784 | 0.217 |
| 4 | 0.617 | 0.095 | 0.011 | 0.298 | 0.362 | 0.003 | 0.176 | 0.163 | 0.379 |
| 5 | 0.359 | 0.088 | 0.000 | 0.671 | 0.988 | 0.060 | 0.041 | 0.485 | 0.067 |
| 6 | 0.000 | 0.061 | 0.004 | 0.303 | 0.000 | 0.471 | 0.237 | 0.083 | 0.101 |
| 7 | 0.794 | 0.029 | 0.001 | 0.839 | 0.006 | 0.024 | 0.001 | 0.301 | 0.810 |
| Standard deviation of RTT | | | | | | | | | |
| 0 | 0.281 | 0.735 | 0.570 | 0.835 | 0.083 | 0.073 | 0.002 | 0.033 | 0.347 |
| 1 | 0.653 | 0.347 | 0.000 | 0.969 | 0.001 | 0.150 | 0.066 | 0.208 | 0.405 |
| 2 | 0.409 | 0.936 | 0.007 | 0.001 | 0.001 | 0.068 | 0.118 | 0.018 | 0.007 |
| 3 | 0.178 | 0.049 | 0.088 | 0.848 | 0.002 | 0.148 | 0.181 | 0.936 | 0.109 |
| 4 | 0.983 | 0.002 | 0.120 | 0.071 | 0.033 | 0.000 | 0.466 | 0.608 | 0.716 |
| 5 | 0.013 | 0.537 | 0.001 | 0.170 | 0.818 | 0.059 | 0.010 | 0.662 | 0.000 |
| 6 | 0.047 | 0.050 | 0.002 | 0.445 | 0.000 | 0.345 | 0.000 | 0.023 | 0.461 |
| 7 | 0.830 | 0.232 | 0.235 | 0.107 | 0.001 | 0.008 | 0.524 | 0.716 | 0.506 |

**Table 3.5** $p$ values for Levene's test of homogeneity of variances.

| Topology | Throughput | Average RTT | St. Deviation of RTT |
|---|---|---|---|
| 1hop | 0.000 | 0.000 | 0.000 |
| 2hops | 0.451 | 0.000 | 0.000 |
| 3hops | 0.760 | 0.000 | 0.000 |
| 4hops | 0.766 | 0.000 | 0.000 |
| 5hops | 0.362 | 0.000 | 0.000 |
| 6hops | 0.240 | 0.000 | 0.000 |
| 2by2 | 0.009 | 0.000 | 0.000 |
| 3by2 | 0.000 | 0.000 | 0.000 |
| 4by2 | 0.070 | 0.001 | 0.000 |

When the group variances are not homogeneous, Dunnett's T3 test is recommended. However, Dunnett's T3 is only a pairwise comparison test and not a range test. For our data however we need a range test so that we can identify subsets of means that do not differ. We want such subsets for all our response variables, so that we can identify the most effective treatments. However, there is no range test available when the group variances are not homogeneous. Scheffe's test, which is both a pairwise as well as range test, seems like the most appropriate choice, since it is somewhat robust to both nonnormality of the response, and heterogeneity of variances among groups. The results from Scheffe's tests are shown in the appendix in Tables A.1–A.3.

### 3.5.4 Results

To interpret the results from Scheffe's multiple comparison test we make another table, Table 3.7, which lists the best homogeneous subset of treatments for each topology and each response variable. We see that the treatments 1, 2, 5, and 6 appear in the best homogeneous subset for all the topologies for the response variables: RTT average and standard deviation of RTT. In fact, Tables A.2 and A.3 reveal that they are far better than the rest, with delays and delay jitters orders of magnitude lower. Thus, we rule out treatments 0, 3, 4, and 7 from possible recommendations. With respect to the throughput, the division into homogeneous subsets is not that clean for all the topologies. In fact, the difference between the means for throughput is not much for different treatments, possibly because we are already operating close to "capacity" for our topologies. The real gains to be reaped are in average delays and delay jitter. For throughput, Table A.1 reveals that the subsets containing treatments 1 and 5 are statistically better than the subsets containing treatments 2 and 6, for all the topologies. This leaves us with treatments 1 and 5, and our experiments do not demonstrate any significant difference between them. Treatment 1 includes window clamping, RTS-CTS handshake off and TCP SACK option on. Treatment 5 is the same as treatment 1 except that SACK is turned off. Thus, our experiments show that there is no statistically significant impact of turning on SACK. This may be because SACK does not really kick in since the MAC retries shield TCP from end-to-end loss. But since turning on SACK does not degrade performance, we recommend turning it on whenever possible.

**Table 3.6** $p$ values for ANOVA tests.

| Topology | Throughput | Average RTT | St. Deviation of RTT |
|---|---|---|---|
| Standard ANOVA $F$ test | | | |
| 1 hop | 0.000 | 0.000 | 0.000 |
| 2 hops | 0.000 | 0.000 | 0.000 |
| 3 hops | 0.000 | 0.000 | 0.000 |
| 4 hops | 0.000 | 0.000 | 0.000 |
| 5 hops | 0.000 | 0.000 | 0.000 |
| 6 hops | 0.000 | 0.000 | 0.000 |
| 2by2 hops | 0.000 | 0.000 | 0.000 |
| 3by2 hops | 0.000 | 0.000 | 0.000 |
| 4by2 hops | 0.005 | 0.000 | 0.000 |
| Welch test | | | |
| 1 hop | 0.000 | 0.000 | 0.000 |
| 2 hops | 0.000 | 0.000 | 0.000 |
| 3 hops | 0.000 | 0.000 | 0.000 |
| 4 hops | 0.000 | 0.000 | 0.000 |
| 5 hops | 0.000 | 0.000 | 0.000 |
| 6 hops | 0.000 | 0.000 | 0.000 |
| 2by2 hops | 0.000 | 0.000 | 0.000 |
| 3by2 hops | 0.000 | 0.000 | 0.000 |
| 4by2 hops | 0.024 | 0.000 | 0.000 |
| Brown Forsythe test | | | |
| 1 hop | 0.000 | 0.000 | 0.000 |
| 2 hops | 0.000 | 0.000 | 0.000 |
| 3 hops | 0.000 | 0.000 | 0.000 |
| 4 hops | 0.000 | 0.000 | 0.000 |
| 5 hops | 0.000 | 0.000 | 0.000 |
| 6 hops | 0.000 | 0.000 | 0.000 |
| 2by2 hops | 0.000 | 0.000 | 0.000 |
| 3by2 hops | 0.000 | 0.000 | 0.000 |
| 4by2 hops | 0.006 | 0.000 | 0.000 |

**Table 3.7** The best homogenous subset for each response variable, obtained from Scheffe's test.

| Topology | Throughput | Average RTT | St. Deviation of RTT |
|---|---|---|---|
| 1 hop | {7, 3} | {1, 5} | {2, 6, 1, 5} |
| 2 hops | {7} | {5, 1, 6, 2} | {1, 6, 2} |
| 3 hops | {7, 1, 5, 3, 2} | {1, 5, 2, 6} | {5, 1, 2, 6} |
| 4 hops | {7, 5, 3} | {1, 5, 6, 2} | {1, 5, 6, 2} |
| 5 hops | {5, 7, 1} | {5, 1} | {5, 1, 6, 2} |
| 6 hops | {1, 7, 5, 4} | {5, 1, 6, 2, 3} | {5, 1, 6, 2, 7, 0, 3 } |
| 2by2 hops | {7, 3, 4, 1} | {1, 5, 2, 6} | {1, 5, 2, 6} |
| 3by2 hops | {7, 3} | {5, 1, 2, 6} | {1, 5, 2, 6} |
| 4by2 hops | {7, 3, 5, 1, 4, 0, 2, 6} | {5, 1, 2, 6} | {5, 1, 6, 2} |

Thus, our final recommendation based on the experimental and statistical analysis is treatment 1. Treatment 1 provides considerable improvements in mean delay and delay jitter, and small but statistically significant improvement in throughput. To conclusively demonstrate this, we display the percent change, with respect to treatment 1, in the means of all the three response variables (displayed in Table 3.3) when using another treatment. For example, the percent change for throughput for treatment $x$ is calculated as (throughput($x$)-throughput(1))*100/throughput(1). These values are shown in Table 3.8.

The entries in bold indicate that the improvement is statistically significant at an *alpha*-level of 0.05, as computed by Dunnett's T3 multiple comparison test. We note that using other treatments can result in orders of magnitude increase in delays and delay jitter compared to treatment 1, and a small but statistically significant decrease in throughput. Thus, our recommendation is treatment 1. It includes setting the congestion window clamp to $\lceil \frac{3}{2}n \rceil$ where n is the number of hops to be traversed by that flow, turning off the RTS-CTS handshake and turning the TCP selective acknowledgment option on.

## 3.6 Concluding Remarks

We have performed a thorough experimental study of the TCP protocol over static wireless multihop networks. The study has led to the identification of several issues and suggested modifications to TCP which are then evaluated through extensive experimentation. The results have been analyzed through statistical tests, and the recommendations made are demonstrated to improve performance. In the process we have discovered numerous subtleties in experimenting with commercial off-the-shelf equipment. Several difficulties had to be overcome before the process of experimentation could be made smooth and tolerably repeatable. We have taken care to collect all possible data and also record the experimental conditions and parameters. The data has been carefully archived and made publicly available [37] through a CGI interface which allows online processing too. More remains to be done, both in terms of analysis as well as experimentation. It is hoped that this study is only the beginning of much needed detailed experimental studies in wireless ad hoc networks.

**Table 3.8** Percentage change in response variables for all treatments, with respect to Treatment 1.

| Treatment | 1hop | 2hop | 3hop | 4hop | 5hop | 6hop | 2by2 | 3by2 | 4by2 |
|---|---|---|---|---|---|---|---|---|---|
| Throughput | | | | | | | | | |
| 0 | **23.7** | -1.46 | **-11.7** | **-8.93** | **-9.30** | **-8.62** | -1.55 | **11.2** | -3.57 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | **-3.90** | **-6.80** | **-6.21** | **-8.93** | **-8.14** | **-10.3** | **-8.25** | **-7.50** | -5.36 |
| 3 | **36.3** | **6.31** | -2.76 | -3.57 | -2.33 | **-8.62** | 5.67 | **15.0** | 9.82 |
| 4 | **28.8** | 0.971 | **-7.59** | **-8.93** | **-9.30** | -8.62 | 1.55 | 7.50 | -1.79 |
| 5 | 0 | 0 | 0 | -0.893 | 4.65 | -5.17 | -2.06 | -0.625 | 1.79 |
| 6 | **-4.50** | **-6.80** | **-8.97** | **-8.04** | **-6.98** | **-10.3** | **-9.79** | **-8.13** | -8.04 |
| 7 | **39.3** | **9.71** | 0 | -0.893 | 3.49 | -1.72 | **8.76** | **21.2** | 8.04 |
| RTT | | | | | | | | | |
| 0 | **701** | **4.29e+03** | **2.41e+03** | **847** | **35.2** | **65.8** | **1.61e+03** | **465** | **127** |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | **5.81** | **8.19** | **8.33** | **9.59** | **6.95** | **6.49** | **12.4** | **8.54** | 4.6 |
| 3 | **357** | **3.92e+03** | **2.23e+03** | **937** | **27.4** | **47.4** | **1.52e+03** | **410** | **120** |
| 4 | **716** | **3.74e+03** | **2.08e+03** | **874** | **39** | **81.8** | **1.72e+03** | **416** | **144** |
| 5 | 0 | 0 | 0.407 | 1.47 | -0.673 | -0.436 | 3.92 | -0.298 | -1.29 |
| 6 | **5.81** | **7.6** | **11.8** | **9.26** | **8.74** | **6.11** | **15.9** | **9.73** | 9.58 |
| 7 | **369** | **3.35e+03** | **2.09e+03** | **758** | **30.2** | **58.8** | **1.55e+03** | **417** | **121** |
| Standard deviation of RTT | | | | | | | | | |
| 0 | **1.91e+03** | **3.64e+03** | **1.82e+03** | **987** | **66.1** | **147** | **1.19e+03** | **578** | **300** |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | -5.13 | 9.09 | 4.37 | 10 | 12.4 | 12.5 | 9.35 | 2.22 | 3.32 |
| 3 | **785** | **3.22e+03** | **1.66e+03** | **1.55e+03** | **70.6** | **148** | **1.08e+03** | **507** | **316** |
| 4 | **1.81e+03** | **3.59e+03** | **1.69e+03** | **1.35e+03** | **69** | 219 | **1.03e+03** | **444** | **301** |
| 5 | 0 | -1.82 | -3.75 | 3.49 | -3.82 | -3.48 | 3.27 | 0.494 | -4.98 |
| 6 | -2.56 | 5.45 | 11.3 | **8.73** | 3.1 | 0.387 | 28.5 | 2.22 | 1.99 |
| 7 | **831** | **2.82e+03** | **1.46e+03** | **1.32e+03** | **68** | **145** | **1.06e+03** | **534** | **308** |

# CHAPTER 4

# SYSTEM SERVICES FOR AD HOC ROUTING

This chapter has a different flavor from the previous two chapters, which dealt with design, analysis, implementation and experimental aspects of power control and TCP. This chapter deals with operating system issues in implementing routing protocols for ad hoc networks. Even though the issues are primarily OS related and quite useful by themselves, the study illustrates how the design of networking protocols impacts their implementation, and hence proliferation, in common operating systems. It gives the important message that architecture and system issues should be given due consideration when designing network protocols.

## 4.1 Motivation

Several routing protocols have been designed and studied through simulations but fewer implementations exist for the testbeds. This chapter[1] focuses on system issues, the goal being to make implementation of ad hoc routing protocols easier. It explores several system issues regarding design and implementation of routing protocols for ad hoc wireless networks. It examines the routing architecture in current operating systems and shows how it is insufficient on several counts, especially for supporting on-demand or reactive routing protocols. Examples include lack of mechanisms for queuing outstanding packets awaiting route discovery, and mechanisms for communicating route usage information from kernel to user-space. This work proposes an architecture and a generic API for any operating system to augment the current routing architecture. Implementing this API, however, requires kernel modifications in general.

[1]The work described here has been done jointly with Yongguang Zhang (HRL Laboratories, LLC) and Binita Gupta.

For the Linux operating system, we provide an implementation of the aforesaid API using a small loadable kernel module and standard facilities like *tun* devices, Netfilter and Raw IP sockets, which are available in the Linux 2.4 kernel. We call this the Ad hoc Support Library (ASL). ASL is provided as a shared user-space library which uses a custom kernel module, but no kernel recompilation is needed. We provide a full-fledged implementation of the AODV protocol using ASL, to demonstrate the viability and usefulness of our framework.

## 4.2 Challenges in Mobile Ad Hoc Routing

This section first outlines the routing architecture in current operating systems and then describes the various challenges in implementing on-demand routing protocols.

### 4.2.1 Current routing architecture

The current Internetworking architecture segregates the routing functionality into two parts: *packet forwarding* and *packet routing* (see Section 4.2 of [85]). Packet forwarding refers to the process of taking a packet, consulting a table (the forwarding table), and sending the packet towards its destination as determined by that table. Packet routing, on the other hand, refers to the process of building the forwarding table. Forwarding is a well-defined process performed locally at each node, whereas routing involves a complex distributed decision making process commonly referred to as the *routing algorithm* or the *routing protocol*. The forwarding table contains enough information to accomplish the forwarding function, whereas the routing table contains information used by the routing algorithm to discover and maintain routes. Strictly speaking, these two tables are different data structures but the two terms are often used interchangeably.

In modern operating systems, packet forwarding is implemented inside the OS kernel whereas routing is implemented in the user space as a daemon program (the routing daemon). Figure 4.1 illustrates the general routing architecture. The forwarding table is inside the kernel and is often called the kernel routing table or route table. Whenever the kernel receives a packet, it consults this table, and forwards the packet to the next hop neighbor through the corresponding

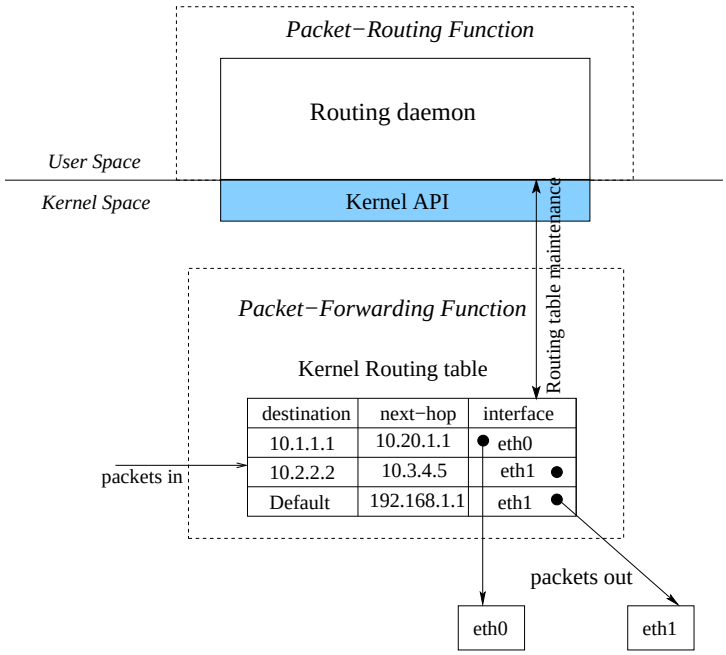



**Figure 4.1** Current routing architecture.

network interface. The entries in the kernel routing table are populated by the routing daemon according to the routing algorithm it implements.

The forwarding-routing separation is in accordance with the mechanism-policy separation of the basic Unix philosophy [86]. The mechanism is efficient and lightweight while the policy is designed to be able to change at a faster time scale without affecting the mechanism. Packet forwarding is the mechanism that must make decisions for every packet and should be efficient. It should reside inside the kernel so that a packet can traverse this node as fast as possible. On the other hand, packet routing is a policy that involves possibly complex CPU/memory intensive tasks, which are properly situated outside the kernel. This principle of separation has made routing in modern operating systems efficient and flexible. It allows the routing function to continue evolving and improving without changing the OS kernel.

### 4.2.2 Challenges in on-demand routing

It is desirable to implement mobile ad hoc routing protocols such that they respect the current routing architecture. Most ad hoc routing protocols can be classified into two cate-

gories: proactive and reactive protocols. Proactive (or table-driven) routing protocols maintain routes to all possible destinations by periodically exchanging control messages. Reactive (or on-demand) protocols, on the other hand, discover routes only when there is a demand for it. Proactive protocols (such as DSDV [87]) can be easily implemented as user-space routing daemons in the current routing architecture in much the same way as routing protocols of the wired world (such as RIP, OSPF, or BGP). However, problems arise with reactive or on-demand routing protocols such as AODV [23] and DSR [24]. We now describe these challenges in detail.

#### 4.2.2.1 Handling outstanding packets

Normally each packet traversing the packet forwarding function will be matched against the kernel routing table. If no entry matches its destination, the kernel will drop the packet immediately. However, this is not a desirable behavior for on-demand ad hoc routing. In on-demand ad hoc routing, not all routes will exist *a priori*; some must be discovered when needed [24]. The correct behavior is to notify the ad hoc routing daemon of a route request and withhold the packet until the discovery finishes and forwarding table is updated. Unfortunately, there is no mechanism in modern operating systems to support this new packet forwarding behavior, and there is insufficient kernel support to implement tasks like queuing of all outstanding packets. Therefore, the operating system should implement the following functionalities for on-demand ad hoc routing:

1. Identify the need for a new route discovery.
2. Notify ad hoc routing daemon of this need for a route.
3. Queue outstanding packets waiting for route discovery.
4. Re-inject them after successful route discovery.

#### 4.2.2.2 Updating the route cache

On-demand routing protocols typically maintain a cache of recently used routes in user space to optimize the route discovery overhead. Each entry in this route cache has an expiration timer, which needs to be reset when the corresponding route is used. The entry should be deleted

(both from the user-space route cache and the kernel routing table) when the timer for that entry expires. Therefore, when an entry in the kernel routing table remains unused (i.e., has not been looked up) for a predefined time period, this information should be accessible to the user-space routing daemon. This is difficult to achieve under the current routing architecture, because no record of route usage in the kernel is available to user-space programs.

#### 4.2.2.3 Intermixing packet forwarding and routing functionalities

Certain ad hoc routing protocols do not have a clean separation between the forwarding and routing functions in their design. For example DSR [24] requires that every packet (not just routing control packets) carry a special DSR header and be processed by the DSR routing daemon. This strong coupling of routing and forwarding functionalities is inherently incompatible with the current OS routing architecture, and is difficult to implement efficiently. Either the complete routing protocol has to sit in the kernel, or packet forwarding itself has to delegated to the user-space by bypassing the kernel routing table.

In some cases, the principle of separation is violated for subtle optimizations aimed at reducing routing overhead. Such optimizations are usually simple to implement in a simulation environment, but present significant system design challenges for actual implementation in operating systems.

#### 4.2.2.4 New routing models

Some ad hoc routing protocols adopt unconventional routing models such as source routing [24], flow-based forwarding [88], etc. These new routing models deviate from the current IP routing architecture and present new challenges in system design. For example, in source routing the entire path that a packet should traverse is determined by the origin of the packet and encoded in the packet header, whereas in traditional IP routing the forwarding decision is made hop-by-hop and driven by the local routing tables. In flow-based forwarding each packet carries a flow ID, and each node in the network has a flow table, which maintains the next hop address and other information for each flow ID. The forwarding is driven by table lookup on the flow ID, and routing is the process to establish the flow table in each node.

#### 4.2.2.5 Cross-layer interactions

Many routing protocols use information from other protocol layers to possibly improve performance. For example, information on the quality of the link to each neighbor may be needed for certain routing algorithms. Even when access to such information is available, it is often not standardized across the various hardware and operating systems. Making lower layer information accessible in a standard fashion is important for development of routing protocols. However, one should exercise caution when engaging in cross-layer design. We return to this topic in Chapter 5.

## 4.3 New Architecture and API

This section first proposes a general solution to support on-demand routing in general purpose operating systems, so that ad hoc routing can be easily supported in the future. We propose enhancements to the current packet-forwarding function with the following mechanisms.

We propose to add an additional flag to each kernel routing table entry to denote whether it is an *on-demand* entry, defined as one for which the kernel is prepared to queue packets in case of route unavailability. An on-demand route entry is said to be *deferred* if it has empty next hop or interface fields, meaning that the route is yet to be discovered. Instead of getting dropped in the normal packet forwarding path, packets matching a deferred route will be processed as described in Section 4.2.2.1. We note that it is not necessary to include every possible on-demand destination in the routing table. Flagging a subnet-based route or the default route as on-demand can serve the same purpose.

A new component, called the on-demand routing component (ODRC), should be added to complement the kernel packet-forwarding function and implement the desired on-demand routing functionalities. When it receives a packet for a deferred route, it first notifies the user-space ad hoc routing daemon of a *route request* for the packet's destination. Then it stores the packet in a temporary buffer and waits for the ad hoc routing daemon to return with the route discovery status. Once this process finishes and the corresponding kernel routing table entry

is populated, the stored packets are removed from the temporary buffer and re-injected into packet forwarding.

To address Challenge 4.2.2.2, a timestamp field needs to be added to each route entry to record the last time this entry was used in packet forwarding. This timestamp can be used to delete a stale route that has not been used for a long time.

Finally, we provide an API so that these new mechanisms can be conveniently used in an ad hoc routing daemon program. The API contains the following functions:

- `int route_add(addr_t dest, addr_t next_hop, char *dev);`
  `int route_del(addr_t dest);`

  These basic routines add or delete an on-demand entry from kernel routing table. To add a deferred route entry, specify `next_hop` to be 0. (Here, `addr_t` is a generic type for the network address, such as `unsigned long` for IPv4 address.)

- `int open_route_request();`
  `int read_route_request(int fd, struct route_info *r_info);`

  ODRC notifies the ad hoc routing daemon about the route requests in the form of an asynchronous stream. The first function returns a file descriptor for this stream and the second function fills in information about the route requests in the second argument, `struct route_info*` which is defined as follows :

```
struct route_info {
        addr_t dest;
        addr_t src;
        u_int8_t protocol;
};
```

  This structure contains information about the packet that triggers the route request, which can be useful for some routing daemons. For example a different action may be warranted if the packet was generated locally rather than being forwarded for some other host (which can be deduced from the `src` field). Similarly, some routing protocols may need to know if the route request is for a TCP packet or a UDP packet.

The file descriptor semantics allows the ad hoc routing daemon to use either event-driven or polling strategy. The function `read_route_request()` blocks until the next route request becomes available.

- `int route_discovery_done(addr_t dest, int result);`

  This function informs the ODRC that a route discovery for the given destination has finished and the kernel routing table populated. The result field indicates whether the route discovery was successful or not.

- `int query_route_idle_time(addr_t dest);`

  Given a destination, this function returns the idle time recorded in the kernel routing table for this entry (elapsed time since the last use of the route).

- `int close_route_request(int fd);`

  This function is called by the routing daemon when it no more desires to receive any further route requests. This enables the ODRC to free the memory used up by packets already queued up and close the fd.

Figure 4.2 illustrates our new architecture and its components. The shaded parts are our proposed additions. Implementing this API in a Unix-like modern operating system usually requires some changes to the system internals. The ideal way is to integrate the above mechanisms into the kernel IP stack. This would involve implementing queuing for every deferred route and adding a special file descriptor for route-request stream. Depending on the operating system's kernel facilities and extensibility, this architecture can also be implemented as kernel modules or largely in user-space.

It is debatable whether the ODRC functionality should be implemented outside the kernel or in user-space. The advantage of a user-space approach is that it reduces the kernel complexity and memory usage. If the routing protocol requires prolonged route discovery procedure, it will be compelling to buffer packets outside the kernel. The disadvantage is the need to copy every deferred-routing packet (i.e., packets awaiting route discovery) from kernel to user-space and to re-inject them back to kernel when the routes are ready, but it can be argued that such overhead is insignificant compared with the time and overhead in an average route discovery.

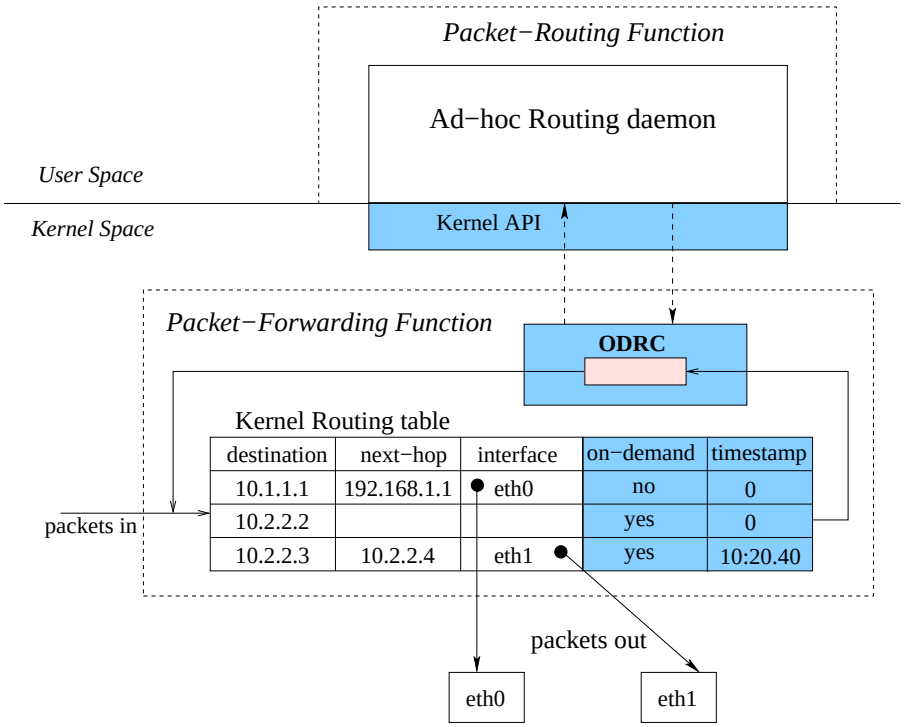



**Figure 4.2** New routing architecture.

## 4.4 Implementation in Linux: Ad Hoc Support Library

We have developed a specific implementation of our purposed API for the Linux operating system. It was possible to implement the whole API without any modifications to the kernel source code. Certain tricks specific to the Linux 2.4 kernel like virtual *tun* devices, raw sockets, Netfilter, etc., were used to achieve this. We now describe our design and implementation, providing background detail as we go along.

### 4.4.1 Design and mechanisms

We place the ODRC function in user-space to reduce the kernel complexity and memory requirements. There are two possible ways to implement this. One approach is to put ODRC in a shared library and link any routing daemon that wishes to use this ODRC function with this library. Another approach is to put ODRC in a separate daemon program and let it communicate with the routing daemon using some interprocess communication mechanism like sockets. Both approaches have their pros and cons. The library approach is more efficient

because it does not have the overhead of interprocess communication, but any bug in the library is likely to crash the routing daemon also. The library approach gives a more natural picture of the ODRC functionalities as system services, i.e., the API is available as direct function calls once the appropriate header files are included.

This work implements the ODRC as a user-space library called ASL. ASL implements the API described in Section 4.3. The implementation consists of two main components: the first component is completely in user-space and implements the common functionalities which are needed by most on-demand routing daemons; the other part is specific to particular routing protocols and is implemented as loadable kernel modules. For example, for the AODV protocol there is the aodv-helper kernel module which provides additional API for some subtle optimizations prescribed by the AODV draft. We also provide a generic helper module called the route-check, which provides a simple solution to the route caching problem. This architecture, consisting of ASL and helper modules, provides the flexibility to incorporate future routing protocols or modifications to current ones in their respective helper modules.

#### 4.4.1.1 Handling outstanding packets

To solve the problem of identifying the need for a route request, we need to filter all packets for which no route exists. Without modifying the routing table structure, there is no simple way to do that in kernel. We solve this by using a local tunnel device called Universal TUN/TAP (*tun*) as the interface device for these destinations. To get packets for all such destinations, we can use the default route which is used for packets that do not match any other entry in the routing table.

TUN/TAP is a virtual tunnel device that makes available all received packets to a user-space program through the /dev/net/tun device. In our implementation, this device is opened by a call to `open_route_request()`; hence, it receives all packets that kernel writes to *tun*, i.e., all packets for which there is no route. This also solves the problem of passing and storing packets in user-space.

Whenever a new packet is read from the virtual device, /dev/net/tun, the ad hoc routing daemon which has opened a route request gets notified on that file descriptor. It can read

the details of the route request through `read_route_request()`, and can then initiate route discovery for the requested destinations. These packets are temporarily queued in a hash table of queues, keyed by the destination IP address. This functionality is implemented in ASL. Since the buffer is in user-space, a large buffer is available to queue packets. This means that packets would not be lost even if the route discovery delays are large.

The next issue is to re-inject packets back into the IP stack after a successful route discovery. The mechanism we use is a raw IP socket.[2] A packet sent through a raw socket is inserted as is (bypassing any IP and header processing) to the kernel output chain just before the packet-forwarding function. Here we use a raw socket to send the queued packets out. These packets are appropriately routed in the kernel using the newly discovered routes.

A natural question to ask is that why do we not re-inject the packets back into the IP stack by writing it on the user end of the same virtual interface used earlier, i.e., /dev/net/tun. To the kernel it appears as if a packet has been received on the *tun* virtual interface, and it can do the routing as if it were a normal incoming packet. This approach works fine for packets which a node forwards, but unfortunately does not work for packets generated locally by the node. Packets which are generated locally already pass through the IP output routines, and when re-injected through *tun* appear on the forwarding chain. The forwarding chain does not allow packets in which the source IP address matches the local IP address since this is an indication that the node's IP address is being spoofed by somebody else. Hence we have to resort to raw IP sockets as described above.

#### 4.4.1.2 Updating the route cache

We now address the problem in Section 4.2.2.2 – to refresh entries in the user-space route cache when a route is used in the kernel. Since we are not making changes to the kernel routing table, the only way is to maintain a separate timestamp outside the kernel for each entry in the routing table. We design a simple kernel module called `route_check` to maintain this table using the Netfilter [89] packet filtering framework. Netfilter provides a set of hooks in the kernel networking stack where kernel modules can register callback functions, and allows

[2]Raw sockets are normally used to handle packets that the kernel does not support explicitly. The `ping` program, for example, uses raw sockets to generate ICMP packets.

them to mangle each packet traversing the corresponding hooks. The `route_check` module is registered at Netfilter's `POST_ROUTING` hook (after routing table lookup and before entering the physical network interface). This means that every outgoing packet will pass through this module. It simply peeks at the packet header and updates the corresponding timestamp value. This timestamp information is made available to user-space programs using an entry in the `/proc` file system. The `query_route_idle_time()` function exposed by the ASL API reads this file (`/proc/asl/route_check`) to determine the idle time for a destination. The routing daemon can check the freshness of a route by reading this file, and delete the stale routes from the kernel routing table accordingly. The route_check module is a generic helper module, which is available to all the routing daemons.

Actually, the current Linux kernel does maintain a cache of most frequently used routes to make routing lookups efficient. When a route is first used it is looked up from the forwarding information base (FIB) which is a complex data structure maintaining all the routes. After first use, this entry is inserted in the route cache for fast lookup. It expires from the cache if not used for some length of time. Information about this route cache is exposed through the files `/proc/net/rt_cache` and `/proc/net/rt_cache_stat`. Unfortunately these files do not include information about the `last_use_time` of the entries. It is a very simple modification to the Linux kernel to make it output this information, but we do want to make any changes at all in the core kernel source as it would require kernel recompilation. Thus, we have adopted the `route_check` module approach just described. We emphasize that a very small change in the Linux kernel would make the route_check module unnecessary.

### 4.4.2 ASL implementation details

Figure 4.3 illustrates the structure of this implementation. The two main components are the user-space library ASL and the kernel module `route_check`. The library implements the API described in Section 4.3. We now describe how we implement these functions in our library.

The `route_add()` and `route_del()` functions add or delete routes to the kernel using the `ioctl()` interface. When the user indicates that the route be a deferred route by specifying an empty next hop, the device for the route is made to be *tun*. The `open_route_request()` function initializes the *tun* device, the raw socket, the data structures to queue deferred packets, and

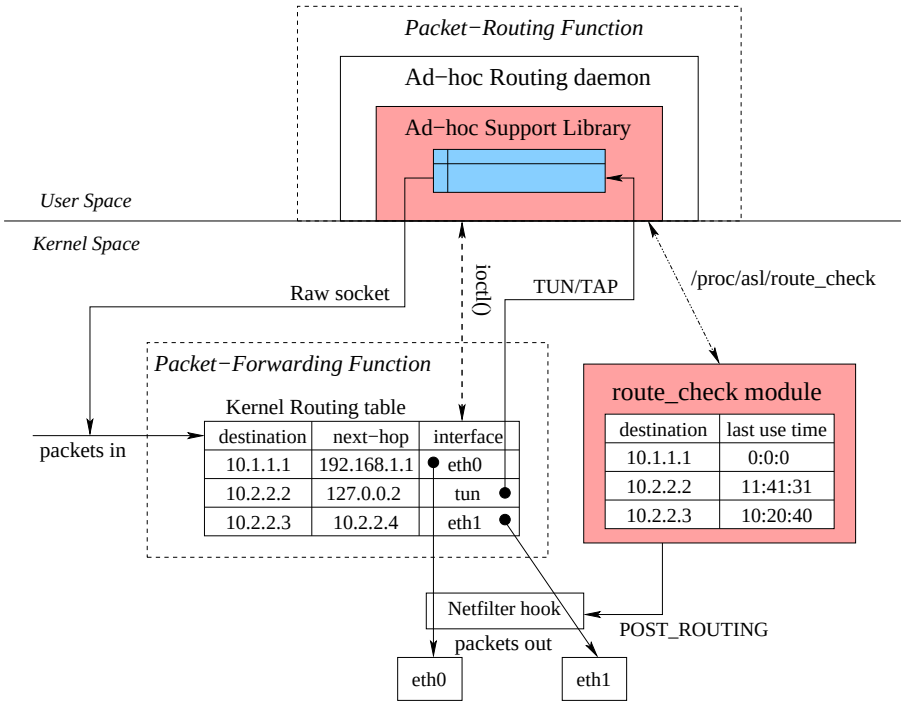



**Figure 4.3** ASL software architecture.

also inserts the `route_check` module in the kernel. The data structure to store the packets is a hash table of queues, keyed by the destination IP address. The function `open_route_request()` returns the descriptor of the *tun* device which can be monitored using a polling or event driven strategy. The `read_route_request()` function blocks reading from this *tun* device. When a packet is received on *tun*, this functions stores the packet and delivers information about the packet in the form of `struct route_info`. Based on this the routing daemon initiates route discovery, and calls the `route_discovery_done()` function on completion of this process. If the route discovery was successful, then this function retrieves the packets for that destination from the storage and sends them out on the raw socket. If a route could not be found then the packets are thrown away and the memory used for them is freed. The `query_route_idle_time()` function reads the `last_use_time` for that destination from `/proc/asl/route_check` and returns the idle time. This needs to be called whenever the routing daemon has to make a decision to expire routes from its user-space route cache. The function `close_route_request()` simply shuts down all the sockets, frees all the memory for storing the packets, and removes the

route_check module from the kernel. Below we give the pseudo code of an example routing daemon which uses this library:

```
aslfd = open_route_request()
route_add(default,0) /* add deferred route */
loop  /* this could be select or poll */
  wait for input from {aslfd or other fd's}
  if input from aslfd
    dest = read_route_request()
    if(route request is new)
      do route discovery for dest
      if successful
        add route for dest to kernel
        route_discovery_done(success)
      else
        route_discovery_done(failure)
      end
    else
      continue
    end
  end
  if input from other fd's
    process according to protocol semantics
    /*call before expiring routes*/
    query_route_idle_time()
  end
end
close_route_request()
```

## 4.5 Implementing Routing Protocols: Experiences using ASL

To demonstrate the flexibility of this approach, a full-fledged implementation of the AODV protocol has been done using ASL. DSR is not as easy to implement since it intermingles packet forwarding and routing to such an extent that it requires a special DSR header in every packet. This amounts to changing the architecture of the network stack, since it requires every packet to be processed by the routing functionality. A "simple" DSR, i.e., source routing in which routes are filled dynamically instead of the user specifying them in every packet, can be implemented easily using ASL. However, respecting all the optimizations prescribed by the DSR draft appears to be a difficult task. For a list of other routing protocols which have been implemented using ASL, please see the project web-page at http://aslib.sourceforge.net.

## 4.6 Existing Implementations and Related Work

There have been some implementations of some on-demand ad hoc routing protocols. These implementations address some or all of the on-demand routing problems, but very few attempt to provide a general framework as we do. In this section, we provide a comparison of how these implementations attempt to address the problems we described in Section 4.2, and suggest how our approach can help improving them.

An implementation study of AODV routing protocol [90] raises issues similar to what we have discussed here. To address on-demand routing problems for AODV, it suggests significant modifications to the existing kernel code. First, IP layer builds a short lived dummy routing table entry for every unroutable destination. It then uses netlink socket to inform AODV routing daemon about the need to initiate a route discovery. Data packets are buffered inside the kernel in a simple linked list referenced from the dummy routing table entry. IP is also modified to add a "last use" field for every route, which is used by the routing daemon when deleting the routes. Our independent investigations led to the identification of similar issues and development of the API we presented in Section 4.3. However, instead of modifying the Linux kernel, we focused on providing a user-space implementation in the form of a shared library, which we hope will be of more immediate use in implementing ad hoc routing protocols.

Madhoc is a user-space implementation of AODV [23]. To address the on-demand routing problem, it snoops ARP packets and uses them as an indication that the destination has no route and route discovery should be triggered. This scheme has a few serious drawbacks. First, the kernel generates an ARP request only if the destination belongs to the subnet of one of the network interfaces, or a host-specific route entry exists for this destination. This limits the applications to certain types of network configurations. Secondly, ARP will time out in relatively short time, and mad hoc provides no mechanism to queue outstanding packets. This means that these packets might be dropped before the route discovery can be completed. Finally, ARP cache has a time-out value and snooping on ARP requests can result in spurious route requests when a next hop node has been timed out in the ARP cache even though the route is still valid.

AODV-UU [91] and AODV-UCSB [92] are two implementations of the AODV routing protocol. The kernel interaction part of the two implementations is the same. They differ only in the AODV protocol logic implementation, which is done in user-space. The kernel part consists of two Linux kernel modules (`kaodv` and `ip_queue_aodv`). To address the on-demand routing problems, these implementations use Netfilter to copy all packets from the kernel space to user-space. The `kaodv` module uses Netfilter to collect all packets before they enter the packet-forwarding function and `ip_queue_aodv` queues them to user-space. By matching these packets against the entries in the user-space route cache, packets for which there is no route can be identified and route request initiated. There are two obvious drawbacks of this approach: every packet has to cross the user kernel address space twice, inducing much overhead, and for every packet the routing is done twice as well, once in user-space and once again in the kernel. In our AODV-UIUC implementation, all such system interactions are cleanly taken care of by ASL. The overhead of processing every packet in user-space is eliminated. The result is a much simpler, cleaner, and more efficient implementation.

Kernel-AODV [93] is another AODV implementation by National Institute of Standards and Technology (NIST). The entire implementation is in the form of Linux kernel modules. As we have discussed earlier in Section 4.2, it is not a good system design to put the entire routing protocol in kernel-space. The complex protocol processing can slow the kernel, hog the memory, and crash the whole system if there are any bugs in the routing protocol.

Another kernel implementation of AODV [94] has been done by extending ARP. Note that unlike Mad hoc which uses the ARP mechanism just to detect route requests, this approach reuses the ARP code in the kernel to do a complete implementation of the AODV protocol. Modified ARP requests and replies are used to generate AODV RREQ and RREP packets. The ARP table essentially acts as the AODV routing table. Modified ARP reply with a special flag is used to generate RERR messages. Data packets are buffered inside the kernel in an ARP queue. This approach is a clever idea, but has the limitations of an in-kernel implementations which we have discussed above.

DSR [24, 95] has been implemented by the Rice University Monarch project in FreeBSD [96]. This implementation is entirely in-kernel and does extensive modifications in the kernel IP stack. While the implementation is a commendable project, it is difficult to maintain and update due to its complexity. Porting it to other operating systems is also difficult.

Internet Manet Encapsulation Layer (IMEP) [97] is an encapsulation protocol proposed for manet routing, which provides certain common functions like single-interface abstraction, link status sensing, control message aggregation, and reliable broadcasting. IMEP attempts to provide a unified framework for all other ad hoc routing protocols at the protocol processing level, whereas we attempt to provide a common module and a common interface at the implementation level. These two approaches are complementary. A simple user-space implementation of IMEP is possible using our framework. Additional support may be needed from the network device driver for link status sensing.

TORA [98] has been implemented in Linux by University of Maryland. It has been implemented over IMEP and hence can benefit from our framework. TORA can be implemented over our framework too.

The University of Colorado has implemented several routing protocols [99–101] using MIT's Click Modular Router [102]. Click is a software architecture for building flexible and configurable routers. It provides basic protocol processing modules called elements, each of which implements a specific function like packet classification, queuing, and scheduling. Protocol developers write Click configuration files to instantiate elements and to connect them in a way to implement the protocol. In this aspect, Click can be a good candidate to meet Challenge 4.2.2.4.

Nevertheless, our work focuses on common operating systems, whereas Click is designed for a special purpose application (fast and configurable routers).

A recent work [103] aims to provide a library of utilities for manet routing protocols, much like the GNU Zebra project does for wired routing protocols. It plans to provide utilities for timer management, neighbor discovery, managing tables etc. It however does not systematically deal with all the system issues, as we have done. ASL could be used in building this framework.

MagnetOS [104] is a distributed, power-aware, adaptive operating system which abstracts an ad hoc network as a unified Java Virtual Machine. It allows for objects and components to automatically migrate among nodes in the network to optimize system performance. MagnetOS focuses on sensor networks, using distributed operating system techniques.

## 4.7 Conclusions

This chapter has studied the operating system services for mobile ad hoc routing, and proposed a generic architecture and API for implementing ad hoc routing protocols in modern OS. We have implemented these services and API in Linux. With the help of standard Linux primitives, we were able to do so without any modification to the core kernel source, i.e., without the need to recompile the kernel. The software consists of a user-space library (ASL) and protocol dependent loadable kernel modules. To demonstrate the flexibility of this approach, we have implement AODV using ASL.

We believe that the principle of separating routing and forwarding has profound importance in ad hoc routing system design. Protocols that follow this principle are easier to implement in a clean way. The code will be efficient, portable, maintainable, and extensible. Protocols that violate this principle and mix routing and forwarding will require significantly more efforts to produce a clean and well-structured implementation. Unfortunately, little attention is paid in today's ad hoc network research to the considerations of system issues in protocol design. Even if the protocol architecture follows the principle of separation, many protocols still suggest subtle optimizations that violate this rule. These optimizations are often easy to simulate but very difficult to implement in real systems. In some cases the complications encountered can

nullify the benefits of the intended optimizations. Through this study, we illustrate that it is important to consider system issues in ad hoc routing protocol design.

# CHAPTER 5

# A CAUTIONARY PERSPECTIVE ON CROSS-LAYER DESIGN

Chapter 4 illustrated how design of networking protocols can affect their implementation in operating systems. This chapter considers the issues of design more explicitly, by addressing the problem of cross-layer design. Cross-layer protocols rely on interactions between different layers. Cross-layer design is popular because it can lead to performance improvements, especially over the wireless channel. However such cross-layer design can run at cross purposes with sound and longer term architectural principles, and can lead to various negative consequences. This motivates us to step back and reexamine holistically the issue of cross-layer design and its architectural ramifications.

This work contends that a good architectural design leads to proliferation and longevity, and illustrates it by some historical examples. However the temptation and perhaps even the need to optimize by incorporating cross-layer adaptation cannot be ignored, and so we examine the issues involved. We show that unintended cross-layer interactions can have undesirable consequences on overall system performance and illustrate some of them by studying certain cross-layer schemes loosely based on recent proposals. Moreover, uncoordinated cross-layer design can lead to spaghetti design, which can be difficult to upkeep. This work suggests that one should exercise caution while engaging in cross-layer design.

## 5.1 Introduction

Recently there has been increased interest in protocols for wireless networks which rely on significant interactions between various layers of the network stack. Generically termed *cross-layer design*, many of these proposals are aimed at achieving performance improvements, though, as we will argue, often at the cost of good architectural design. This has prompted us to take a step back and examine the whole issue of cross-layer design and its architectural ramifications, and more generally the issue of an appropriate architecture for wireless networks.

We begin by emphasizing the importance of a good architectural design. Several historical examples demonstrate the proliferation and longevity that a sound architectural design can bring about. John von Neumann's architecture for computer systems, the layered architecture for the Internet, Shannon's architecture for communication systems, and the plant controller feedback paradigm in control systems, are some successful examples.

A significant measure of credit for the Internet revolution can be attributed to its layered architecture, to the extent that it has become the de facto architecture for wireless systems. Wireless systems however are different, and there is no concept of a link. A transmission is just a spatiotemporal footprint of radio energy. Nodes along a path in a wireless network may operate upon this energy in a variety of ways. In addition to the wireline strategy of "decode and forward" [105], there are various other possibilities like "amplify and forward" [106], "co-operative interference cancellation" [105], etc. The first issue is therefore to choose a sound design from the several possible options, and it is gratifying that "decode and forward" is order optimal for wireless networks as well [105]. By *order optimal* we mean that the network capacity achieved by this scheme, as a function of the number of nodes, is as good as any other scheme up to a multiplicative multiplicative constant. We illustrate in Section 5.3 how this naturally translates to the layered architecture being a reasonable first template for designing wireless networks.

However, various optimization opportunities do present themselves through cross layer design, and the temptation cannot be ignored. But one should exercise caution. Once the layering is broken, the luxury of designing a protocol in isolation is lost, and the effect of any single design choice on the whole system needs to be considered. Cross-layer design can create loops,

and it is well known from control theory that stability then becomes a critical issue. Compounding this is the fact that some interactions are not easily foreseen. Cross-layer design can thus potentially work at cross purposes and the "law of unintended consequences" can take over if one is not careful, and a negative effect on system performance is possible. We illustrate this by a few examples loosely based on recently proposed schemes. The first involves an adaptive rate MAC protocol, and the second involves nested adaptation of transmit power.

Design in the presence of interacting dynamics needs a care all of its own. Besides stability there is also the issue of robustness. Techniques such as time-scale separation may need to be employed to separate interactions, and an accompanying theoretical framework may be needed.

Uncoordinated cross-layer design can also lead to a "spaghetti" design. Further design improvements may then become difficult since it will be hard to address satisfactorily how a new modification will interact with already existing dynamics. Second, it will be hard to upkeep. Rather than modifying just one layer, the entire system may need to be replaced.

All of the foregoing is a manifestation of the ever-present tension between performance and architecture. Performance optimization can lead to short term gain, while architecture is usually based on longer term considerations. We argue that the longer term view is paramount, and that greater caution therefore needs to be exercised when indulging in cross-layer design.

## 5.2 The Importance of Architecture

Architecture in system design pertains to breaking down a system into modular components, and systematically specifying the interactions between the components. The importance of architecture is difficult to overemphasize. Modularity provides the abstractions necessary for designers to understand the overall system. It accelerates development of both design and implementation by enabling parallelization of effort. Designers can focus their effort on a particular subsystem with the assurance that the entire system will interoperate. A good architectural design can thus lead to quick proliferation. Moreover, it can lead to massive proliferation. When subsystems are standardized and used across many applications, the per unit cost is reduced, which in turn increases usage. In contrast, a system which does not capitalize on the amortization of effort by exploiting commonality will essentially need to be

hand crafted in every application, resulting in great expense. Architecture also leads to longevity of the system. Individual modules can be upgraded without necessitating a complete system redesign which would stifle the further development and longevity of the system.

On the other hand, taking an architectural short-cut can often lead to a performance gain. Thus there is always a fundamental tension between performance and architecture, and a temptation to violate the architecture. However, architecture can and should also be regarded as performance optimization, though over a longer time horizon. An architecture that allows massive proliferation can lead to very low per-unit cost for a given performance. Therefore the tension can more appropriately be ascribed to realizing short term gains versus longer term gains. In the particular case of wireless networks, our contention is that the longer term view of architecture is paramount.

We begin by illustrating the importance of architecture in the proliferation of technology by considering some important examples.

### 5.2.1 The von Neumann architecture

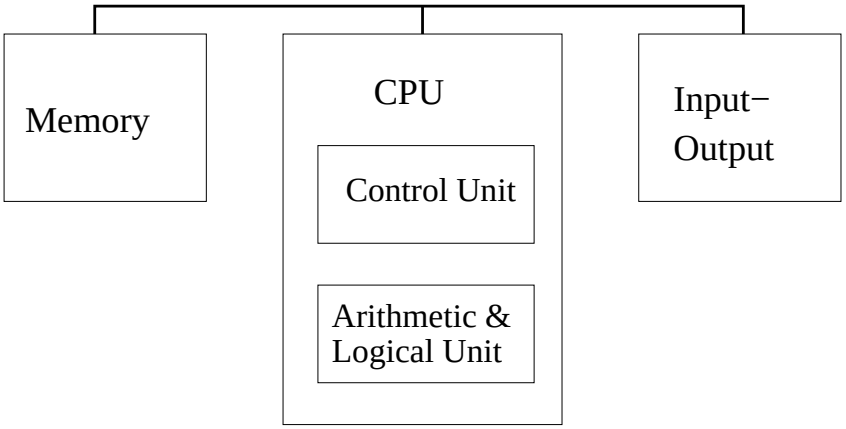



**Figure 5.1** The von Neumann architecture for computer systems.

The von Neumann [107] architecture for computer systems consisting of memory, control unit, arithmetic and logical unit, and input-output devices (see Figure 5.1), is at the heart of most computer systems. It is so much taken for granted now that it needs an effort to appreciate the insight that invented this architecture. It is obvious only in retrospect. Its most important consequence is that it has decoupled software from hardware. Software designers and hardware

designers (e.g., Microsoft and Intel) can work independently, and still be assured that their products will interoperate as long as they each conform to an abstraction of the other side. Software designers can thus develop complex products without worrying about what hardware they will eventually run on. Similarly, hardware designers need not design their hardware for specific application software. Both are in stark contrast to the early ENIAC, for example. As Valiant [108] notes, it is this von Neumann "bridge" that is successful for the proliferation of serial computation. Valiant [108] also points out that it is the lack of such an architecture for parallel computation that is one of the reasons it has not proliferated as successfully. Like any good architecture, the von Neumann architecture has withstood the test of time.

### 5.2.2 The OSI architecture for networking

The layered OSI architecture (Figure 5.2) for networking on which the current Internet architecture (Figure 5.3) is loosely based, is another successful example. It has enabled diverse networks to be interconnected efficiently. The hierarchy of layers provides natural abstractions to deal with the natural hierarchy present in networks, again a fact which may only be obvious with the benefit of hindsight. The physical layer deals with signals, and provides a service to communicate bits. The data link layer provides the abstraction of a link, and the ability to transmit and receive groups of bits, over the link. The network layer introduces the concept of a path or a route, which is a sequence of links. The transport layer provides an end-to-end pipe or channel, which could be reliable or not, depending on the protocol being used. The remaining three layers are less clearly defined, and have been actually merged in the TCP/IP architecture that developed later. The interactions between the layers is controlled, and conducted primarily through protocol headers which each layer prepends to packets.

In addition, the architecture also involves the notion of "peer-to-peer" protocols (such as TCP) which mediate between corresponding layers on different hosts. Given this architectural foundation, designers can afford the luxury of concentrating on designing the protocols for their layer efficiently, with the assurance that the overall system will function reasonably well.

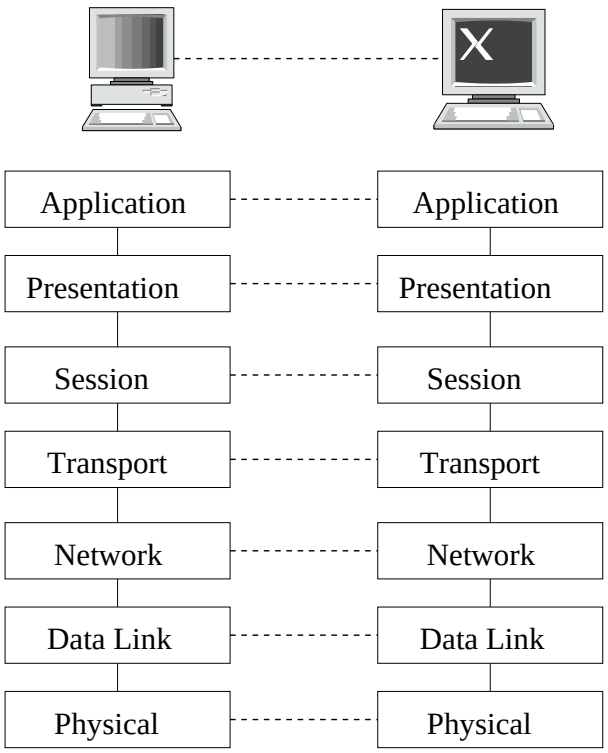



**Figure 5.2** The layered OSI architecture.

### 5.2.3 Source-channel separation and digital communication system architecture

In his seminal work on the problem of information transmission, Shannon and Weaver [109] determined the capacity of discrete memoryless channels as the supremum of the mutual information between channel input and output. More important than this expression for the capacity, we contend, is his constructive proof for an architecture where the "layers" of source compression, and coding for reliability over an error prone channel can be separated, and that the latter layer is invariant with respect to sources. Dubbed the source-channel separation theorem, it allows us to associate source coding (data compression) with the source, and channel coding (error protection) with the channel, thus allowing the same channel plus channel code to serve a large variety of sources, without loss of (near) optimality (see Figure 5.4.) That there is nothing to be gained by designing a code which takes into account the source and the channel statistics simultaneously, is of profound importance and has has great impact. In today's communication systems, source coding is done by a program (gzip or bzip2) in the operating

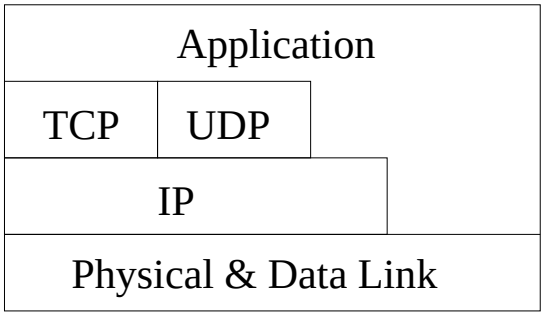



**Figure 5.3** The TCP/IP architecture of the current Internet.

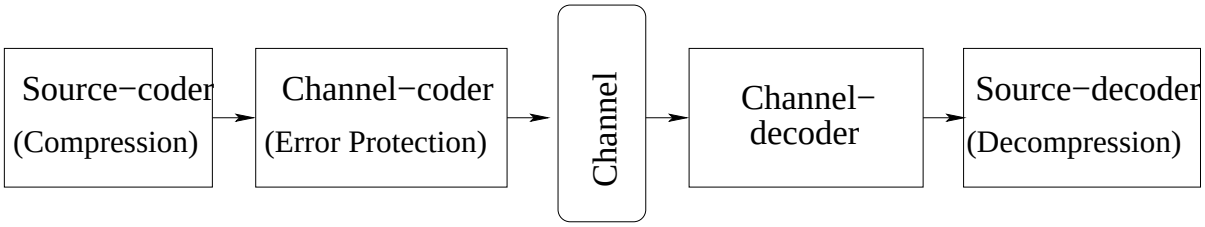



**Figure 5.4** Source-channel separation theorem and the architecture of digital communication systems.

system, whereas channel coding is part of the network interface card designed for a particular channel. It is a huge convenience that the network cards do not have to be designed for a specific application. This architecture has thus had important consequences on the development and proliferation of digital communication systems.

### 5.2.4 Feedback control systems

In control systems, the feedback architecture is widely used. While the design of the feedback layer is not independent of the plant, the architecture of the solution is universal. It is one principle that is common to human engineered systems as well as naturally occurring biological systems.

More generally, the importance of architecture is not confined to engineering. The replacement of the barter system by the currency system decoupled trade from the need to have a buyer-cum-seller at each of the two ends of a transaction, and replaced it with a much more flexible system where one end could be just a buyer and the other just a seller. Currency pro-

vided a bridge between different transactions leading to proliferation of financial transactions and trade.

## 5.3 Architectural Considerations for Wireless Networks

Successful architectural designs also influence how designers think by their pervasiveness. The success of the layered architecture for wired networks has had a great impact on network design paradigms. It has become the default architecture for designing wireless networks as well. However, it is not at all obvious that this architecture is *a priori* appropriate for wireless networks. The reason all this needs reexamination is because the wireless medium allows modalities of communication that are simply nonexistent for wired networks. Our first task is therefore to reexamine from scratch the whole architectural basis for wireless networks. This will allow us to understand where layering stands.

To do this, we first need to be very clear about what is the precise description of the current proposal for ad hoc networks being pursued by IETF and other researchers. We need to be crystal clear not only about the constructive aspect of the scheme, but also what other possible choices have been foreclosed in the current solution. First, packets are transported over several hops. At each hop, the receiver receives not just the intended signal, but also the superposition of the interfering transmissions of other nodes, plus ambient noise. The interfering transmissions are treated as noise, and the packet is fully decoded. This digital regeneration of a packet lies at the heart of the digital (as opposed to analog) revolution. The regenerated packet is then rebroadcast to the next node, where again it is decoded in the presence of noise with other interfering transmissions simply regarded as noise, etc.

There are several reasons why this solution is appealing. First, receivers can afford to be simple, since they merely depend on an adequate signal-to-noise ratio. Second, the "decode and forward" feature along with multihop relaying provide us the abstraction of "links." Thus, protocols for wireline networks can be reused for wireless networks.

However, it should be kept in mind that the choice of a multihop architecture with decoding and forwarding at each relay node while treating all interference as noise, creates certain special problems for wireless networks that need to be solved. For example, since one is treating all

interference as noise, one wants to regulate the number of potential interferers in the vicinity of the receiver. This necessitates a medium access control protocol whereby one controls the number of interferers. This is rendered difficult by the presence of hidden and exposed terminals, and the need to have a distributed real-time solution.

Another problem we have created is the routing problem. This arises because we wish to employ a multihop scheme involving relays. One then needs to find a sequence of relays from origin to destination—the routing problem. Note that there would be no routing problem at all if we were to communicate in just one hop. However, then there would be need for other protocols to distinguish between transmissions which are superimposed.

Yet another problem is the power control problem. Because all interference is regarded as noise, it is useful to regulate the powers of transmitters. This necessitates power control. Power control also arises from the need to maximize capacity by spatial reuse of the spectral frequency resource.

Thus, we see that it is our choice of architecture that has created protocol needs—in particular, medium access control, routing, and power control.

However there are other choices. For example, interference need not interfere. Consider a powerful transmission "interfering" with a weak transmission. Since the powerful transmission has a high SNR, it can be successfully decoded. Therefore it can be subtracted, assuming good channel state information, and the weak signal can be successfully decoded. Thus the more powerful transmission has not "collided" with the weaker one.

### 5.3.1 Fundamental properties of the wireless medium

The first point to note is that there is no intrinsic concept of a link between nodes in a wireless medium. Nodes simply radiate energy, and communicate through the superposition of each others transmissions. There is no notion of a "switch" which allows a receiver to receive just one transmission while shutting out all others. While this could be regarded as a pure liability — and often is by designers who wish to operate it as a sequence of links — it also offers possibilities unimaginable in wireline networks. One really needs to dispel all existing notions and think afresh about designing wireless networks.

### 5.3.2 Several possibilities for operating wireless networks

The fact is that the wireless channel potentially offers other modalities of cooperation, as we next examine. For example, a node A (or a group of nodes) could cancel the interference created by a node B at a node C. Thus, A can help C by reducing the denominator in its signal-to-interference plus noise ratio, rather than by boosting the numerator. At the same time, node A could also expend a portion of its energy to relay packets from node D to node E. Moreover, the relaying could be based on amplifying the received signal, rather than first decoding it, regenerating the packet, and then retransmitting it. Indeed, for some parameter values "amplify and forward" may be preferable to "decode and forward" [106]. Thus there are several possibilities. In fact there need not even be layers. The search space is not finite dimensional, but infinite dimensional. Some recent results however have indicated how wireless networks should be structured, that is, what its architecture should be. Here the focus is on electronic transfer of information rather than mechanically through mobility.

**Theorem 3 (Multihopping Is Order Optimal [105]).** *Suppose there is absorption in the medium, leading to exponential attenuation with distance, or the exponent in polynomial path loss is greater than three. Then multihop decode and forward, treating interference as noise is order optimal with respect to the transport capacity when load across nodes can be balanced by multipath routing. Here, transport capacity is defined as the distance weighted sum of rates. Moreover, the transport capacity grows as* $\Theta(n)$ *(where* $n$ *is the number of nodes), when the nodes are separated by a minimum positive distance.*

In fact, such exponential attenuation is the norm in the scenarios of interest in ad hoc networks [110], rather than the exception. In scenarios with absolutely no exponential attenuation, and low path loss exponent, other strategies emerge as detailed in the theorem below.

**Theorem 4 (Architecture Under Low Attenuation).** *If there is no absorption and the polynomial path loss exponent is small, then other strategies like "coherent multistage relaying with interference subtraction" can be order optimal [105] with respect to the scaling law for transport capacity. The transport capacity can grow super-linearly in* $n$ *in a network of nodes lying along a straight line.*

The first result is very useful. It establishes the order optimality of the multihop decode and forward proposal. It also provides architectural guidance since the multihop decode and forward strategy establishes the need for certain functions in operating a wireless network. Each of these functions can be thought of as a layer in the protocol stack. Decoding at each hop means a data-link layer is needed to deal with a one-hop packet transmission, though there are some important differences from the wired case. For example, the medium access sublayer is usually necessary for the wireless case to solve the channel access problem. Also, due to the high error rates in wireless, a somewhat reliable link layer (i.e., hop-by-hop acknowledgment) is necessary. After decoding, the packet needs to be forwarded, which brings in the concept of a path, and hence the network layer. The transport layer is necessary, as before, to provide an end-to-end reliable pipe. Organizing these functions into layers and operating a wireless network based on such a layered stack is a natural way to implement the multihop decode and forward strategy. Thus a layered architecture can achieve the optimal performance, within a constant, with regards to network capacity.

The above result shows that tweaking with layering can only improve throughput by at most a constant factor—which may still be very important—but they cannot result in any unbounded improvements even in networks with large numbers of nodes [105].

## 5.4 Cross-Layer Design Principles

While we have established that a layered architecture is order optimal and thus a good candidate for a baseline design, there is still the desire, and perhaps a need, to optimize. And indeed several optimization opportunities do present themselves through increased interaction across the layers. There have been numerous cross-layer design proposals which explore a much richer interaction between parameters across-layers [111].

In evaluating these proposals, the trade-off between performance and architecture needs to be considered. As noted above the "performance metrics" of the two are different. The former is more short-term, while the latter is longer term. Thus a particular cross-layer suggestion may yield an improvement in throughput or delay performance. To be weighed against this are longer term considerations.

First, while an individual suggestion for cross-layer design, in isolation, may appear appealing, what is the consequence when other cross-layer interactions are incorporated? Can these interactions work at cross purposes? The point is that every installation of a cross-layer interaction rules out simultaneous use of other potential interactions which might interfere with it. Thus, the case for adoption of a cross-layer interaction must be made more holistically by showing why other mechanisms, which are potentially conflicting with the suggested one, should not be entertained.

Evaluating the merit of proposals on the overall architecture is thus not an easy task. However, there are some general cautions that can be exercised. We make an attempt to derive some general principles to assist in this process.

### 5.4.1 Interactions and unintended consequences

The layered architecture and the controlled interaction enable designers of protocols at a particular layer to work without worrying about the rest of the stack. Once the layers are broken through cross-layer interactions, this luxury is no longer available to the designer. The interaction can affect not only the layers concerned, but also other parts of the system. It is important to consider the effect of that particular interaction on a remote, seemingly unrelated, part of the stack. There could be unintended consequences which may negatively effect overall performance.

### 5.4.2 Time-scale separation and stability

Cross-layer design often causes several adaptation loops which are parts of different protocols to interact with each other. To comprehend possible interactions, it is useful to represent the protocols graphically as interactions between parameters. In this *dependency graph* every relevant parameter is a node, and a directed edge indicates the dependency relation between the parameters. By combining the graphs for various protocols, a dependency graph for the entire stack can be obtained.

Certain stability principles can be derived by observing the dependency graph. If a parameter is controlled and used by two different adaptation loops, then they can conflict with each

other. It is well known from adaptive control theory [112, 113] that such adaptation can benefit from time scale separation. The idea behind it is simple. Consider two entities controlling the same variable but on different time scales. Then by using the notion of averaging and time scale separation, stability theorems have been proved. The slower system can be regarded as seeing the averaged value of the parameter, and its stability implies the stability of the overall system under some conditions [114].

For every closed loop in the dependency graph consisting of interactions at similar timescales, proofs of stability are required. This is often nontrivial and requires significant analytical effort. Designers of cross-layer protocols will need to contend with this.

### 5.4.3 Uncoordinated cross-layer design

In addition to these factors, yet another larger issue needs to be considered. What if several cross-layer interactions are implemented? Does one then get unstructured spaghetti-like code that is hard to maintain? Will the resulting system have longevity? Will the need to update the whole system for every modification stifle proliferation? Will this lead to a higher per-unit cost, which eventually is regarded by the end user as a lower performance value?

## 5.5 Illustration by Examples

To more concretely illustrate the possibility of unintended interactions, we now present some examples by simulation studies. These examples are loosely based on schemes proposed recently in literature, and are expressly not intended to cast any aspersion on the particular schemes themselves. Indeed the only reasons for the choice of the particular examples was that we were able to construct scenarios which exhibit the type of negative results that we want to demonstrate, when they are used in tandem with other existing protocols.

### 5.5.1 Rate adaptive MAC and minimum hop routing

The idea behind rate adaptive MAC protocols is to send data at higher rates when the channel quality is good [115]. Such higher rates are achieved by changing the modulation

scheme. As we will show, such schemes can have undesirable consequences for the higher layers. In particular we show that when combined with minimum hop routing—and most routing protocols are indeed minimum hop— they can lead to performance worse than the original system. Briefly, the reason for the adverse behavior is as follows. Minimum hop routing chooses longer hops, for which the signal strength is lower, and thus the data rate achieved through channel quality adaptation is low.

We are not suggesting that adaptive rate MAC is a bad design. In fact, in this example it may be good and the problem may lie in the use of a min hop routing protocol. Perhaps a routing protocol using some other metric, or a load adaptive routing protocol may be necessary. The whole point we seek to make is that cross-layer design can lead to adverse unintended interactions, and adequate care is necessary.

The protocol details of the specific adaptive rate MAC are as follows. It is a modification of the IEEE 802.11 MAC protocol. There are a set of rates available, i.e., modulation schemes, and the transmission rate can be set before transmitting every packet. The RTS/CTS and broadcast packets are always transmitted at the lowest data rate, called the base rate. The receiver measures the received signal strength of the RTS packet, and figures out the maximum rate at which data can be received given that signal strength. This rate is then communicated to the sender in the CTS packet. The subsequent DATA and ACK packets are transmitted at this data rate.

Typically, transmission to nodes close by would occur at higher data rates since the path loss attenuation of the signal is small, while lower data rates would be used for farther nodes for which the received signal is weaker. A modification to this scheme is to opportunistically send more packets when the channel is good [116]. The idea is to "make more hay while the sun shines." Every reservation allocates a fixed time slot, and if the channel is good, there is the opportunity to send more than one packet at higher data rates. These seem like very reasonable cross-layer designs.

However, consider the interaction with higher layers. Suppose we were to use a minimum hop routing, say a protocol like DSDV [35]. DSDV builds routing tables by sending *hello* packets to neighbors. *Hello* packets are broadcast packets which contain cumulative routing information, that is, information that has been gathered from all the neighbors of a node. Since

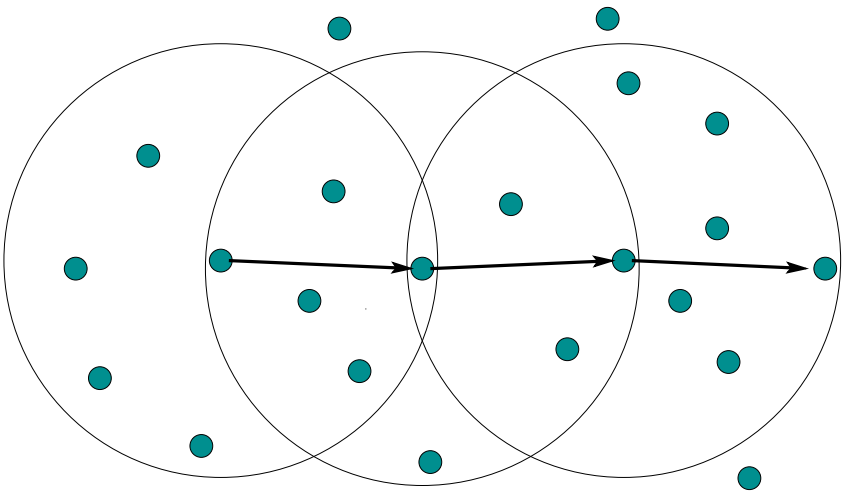


**Figure 5.5** DSDV chooses small number of long hops, which give a lower data rate when an adaptive rate MAC is used.

*hello* packets are broadcast packets, they are sent at the base rate, and thus have a large range. Minimum hop routing thus chooses the longest possible hops on the path, which causes low received signal strength, which in turn implies a low data rate. This is shown in Figure 5.5.

In fact, if we turn off the adaptive rate MAC and use plain IEEE 802.11 at the highest data rate, i.e., do not send any data when the channel is not good enough to transmit at the highest rate, we can get much better end-to-end throughput. In this case, longer hops simply do not exist, and thus min hop routing is forced to use a larger number of short hops, which provide higher data rates, as depicted in Figure 5.6.

#### 5.5.1.1 Verification by NS2 simulations

We now verify the above through NS2 simulations. Scheme 1 uses the adaptive rate MAC with opportunistic scheduling, and our simulation uses, in part, the code provided in [117]. The two-ray-ground propagation model was used for the channel, which results in an $r^{-4}$ attenuation between distance and received signal strength, in the relatively simple interference model used in NS2. Scheme 1 is set up such that at a fixed transmit power level of 0.28 W, a receiver-transmitter distance of 0-99 m yields a data-rate of 11 Mb/s, a distance of 100-198 m yields a data rate of 5.5 Mb/s, while if the distance is between 199-250 m then only 2 Mb/s is possible. No communication is possible beyond 250 m. A simple opportunistic policy which gives equal

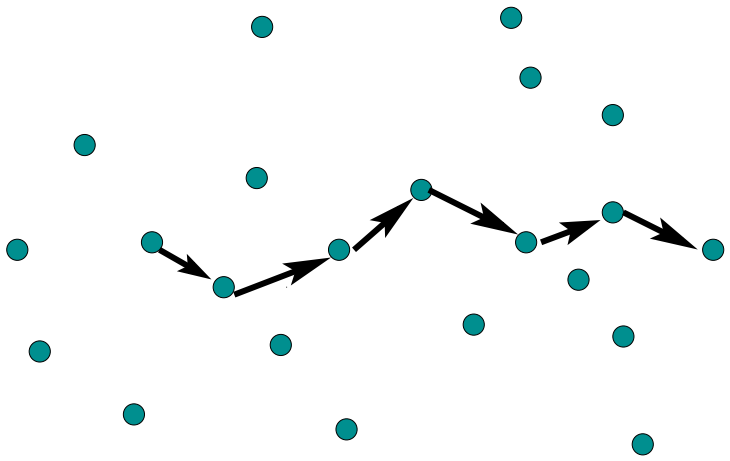


**Figure 5.6** Plain IEEE 802.11 causes short hopes of higher data-rate to be used.

time share to each data rate is also implemented. Thus a max of five packets are transmitted at 11 Mb/s, three at 5.5 Mb/s, and only one packet if the channel is good enough only for 2 Mb/s.

Scheme 2 is plain vanilla IEEE 802.11 with a data-rate of 11 Mb/s. That is, a packet is sent at 11 Mb/s if the receiver-transmitter distance is less than 100 m, else no communication is possible. Thus full use of the channel is not made by transmitting at lower data rates when the channel is not good enough. Carrier sensing was turned off for both Schemes 1 and 2.

For the first experiment, 18 nodes are equally spaced in a straight line, in a 1500 m $\times$ 200 m area. One TCP connection is run across the end of the chain from Node 0 to Node 17. The DSDV routing protocol is used, and a TCP connection is started after the routing tables have stabilized. The received throughput for the two schemes is plotted in Figure 5.7. As predicted, Scheme 2 outperforms Scheme 1. The average end-to-end throughput for Scheme 1 is 226 kb/s, whereas it is 455 kb/s for Scheme 2 which is without any adaptation.

In a second experiment, fifty nodes are randomly located in a 1000 m $\times$ 200 m rectangular area (Figure 5.8). It was ensured that the topology was connected for Scheme 2 as well, i.e., when the transmit range is 100 m. Five TCP connections were then started simultaneously between far away nodes. The DSDV routing protocol was run as before. The results are shown in Figure 5.9. Scheme 2 (i.e., the one without adaptation) again significantly outperforms Scheme 1. The total throughput for all the flows is 349 kb/s for Scheme 1, while it is 583 kb/s for Scheme 2.

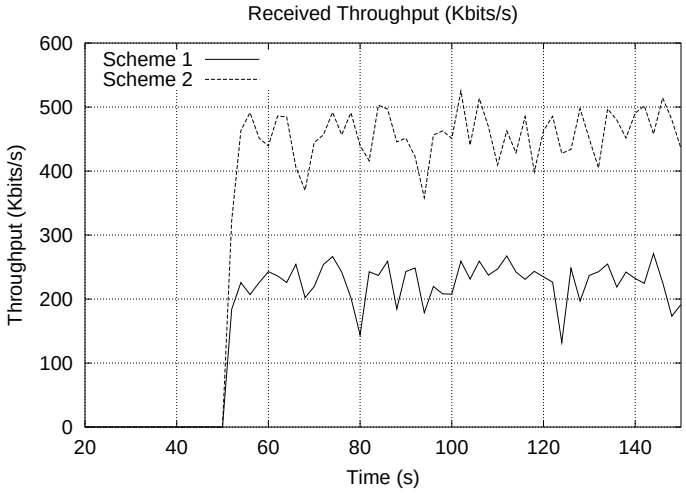



**Figure 5.7** Comparing Scheme 1 (adaptive rate MAC) and Scheme 2 (plain IEEE 802.11) for the 18 node linear topology.

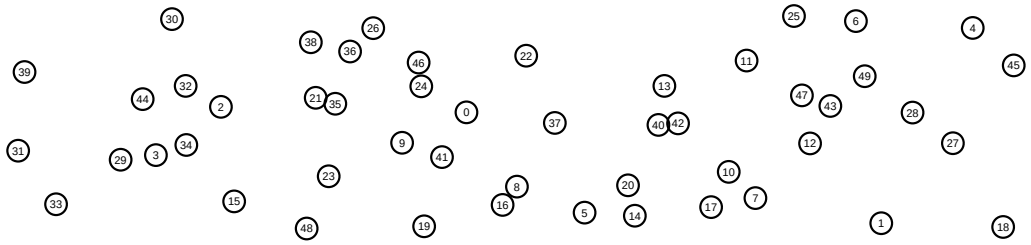


**Figure 5.8** Fifty nodes placed randomly in a 1000 × 200 m area.

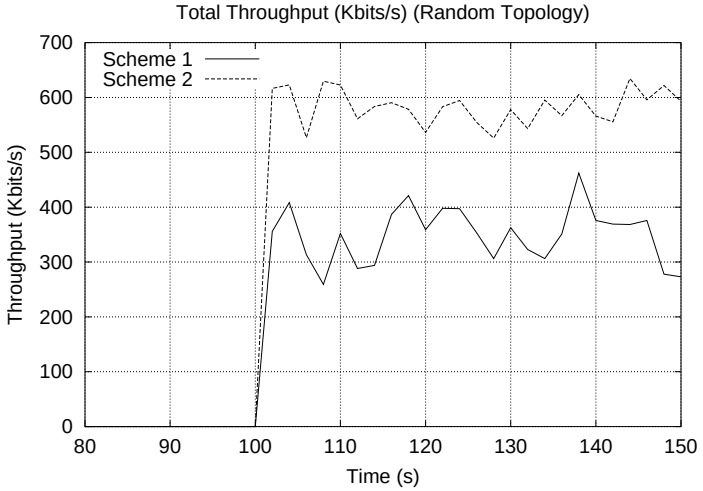



**Figure 5.9** Comparison of adaptive rate MAC (Scheme 1) and plain IEEE 802.11 (Scheme 2) for the topology shown in Figure 5.8.

### 5.5.2 End-to-end feedback and topology control

We next consider another example of cross-layer design to illustrate unintended consequences at other layers. The broad idea of the scheme is to adjust the number of neighbors of each node so that TCP performance is enhanced. This is done in two stages. This scheme is a modification of that suggested in [42], and consists of two nested adaptation loops. In the inner loop, each node controls its transmit power so that the number of one-hop neighbors (*out_degree*) is driven to a parameter called *target_degree*. Transmit power is increased by one level if the number of one hop neighbors is less than *target_degree*, and decreased if it is greater than *target_degree*. There is an outer loop which sets the value of this parameter, *target_degree*, based on the average end-to-end network throughput. The action of the previous iterate is repeated (increase by one, decrease by one, or do nothing to the *target_degree*) if the network throughput increased from the previous iterate, and it is reversed if the network throughput decreased from the previous time step. If the network throughput is zero then the network is assumed to be disconnected and the *target_degree* is increased by one.

The outer loop is operated at a slower time-scale compared to the inner loop to avoid the instability and incoherence that results from two simultaneous adaptation loops interacting with the same phenomenon, as elaborated in Section 5.4.2. It should be noticed that employing just the inner loop alone would be a bad design since it does not even guarantee network connectivity if *target_degree* is set arbitrarily. The network could thus get stuck in an absorbing state of zero throughput. The outer loop therefore attempts to drive *target_degree* to a value which maximizes the end-to-end network throughput, and would therefore eventually drive the network out of any possible state of disconnectivity.

The consequences of the two loop scheme on the performance of higher layers also needs to be examined. As we see below, simulation studies in our deliberately contrived scenarios show that the network may oscillate between connectivity and disconnectivity, which affects TCP performance adversely.

#### 5.5.2.1 Simulation studies

The topology control scheme with the two adaptation loops described above was simulated using the NS2 simulator. In the simulations, the inner adaptation loop adjusts the transmit power level every 15 s to bring the *out_degree* close to the *target_degree*. Each node has 5 transmit power levels, which correspond to transmit ranges of 100 m, 140 m, 180 m, 220 m, and 250 m, respectively, when the two-ray-ground propagation model is used. The outer loop of *target_degree* adaptation is carried out once every 90 s. The topology consisting of 23 nodes in a 500 m $\times$ 500 m area is depicted in Figure 5.10. There is one TCP connection from Node 0 to Node 3.


**Figure 5.10** Simulation topology for cross-layer design involving power control and end-to-end feedback.

Initially the power level of all the nodes is set to the lowest value, and *target_degree* is set to 2. The network is disconnected in this state and we get zero throughput. The *target_degree* is gradually increased according to the outer loop until the power level is high enough for the network to be connected. The action is repeated if the throughput keeps on increasing, but after a while it decreases because of increased interference due to the high power levels. Thus, *target_degree* is decreased and the network goes back into a state of disconnectivity. Consequently, we observe oscillations in the end-to-end throughput, which results in poor average performance over time.

The results from the simulations are shown in Figures 5.11 and 5.12. Figure 5.11 plots the TCP throughput with time, which hits zero periodically as the network gets into a state of disconnection. Figure 5.12 shows the oscillations in the size of the routing table and the transmit power which cause the oscillations in the TCP throughput.

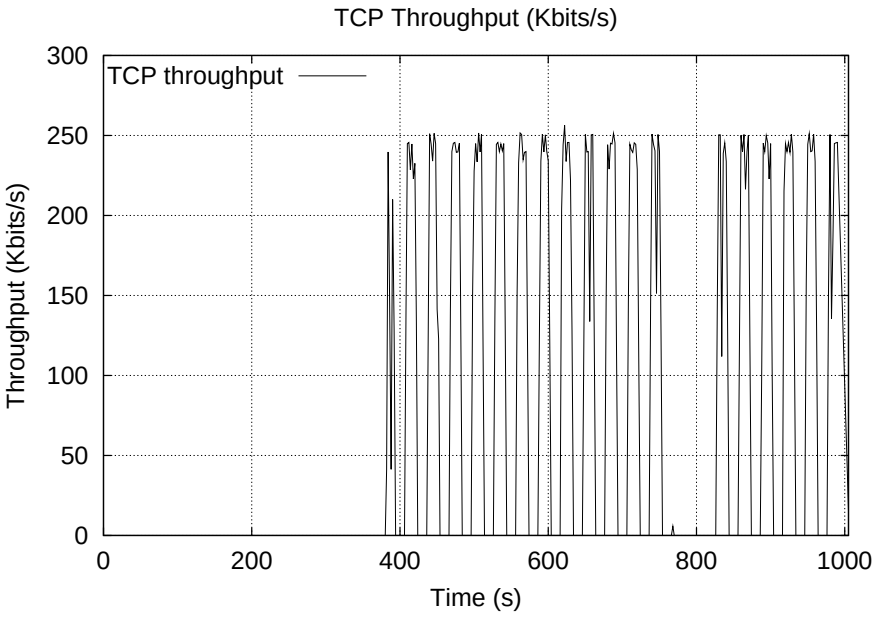



**Figure 5.11** End to end throughput for the topology of Figure 5.10.

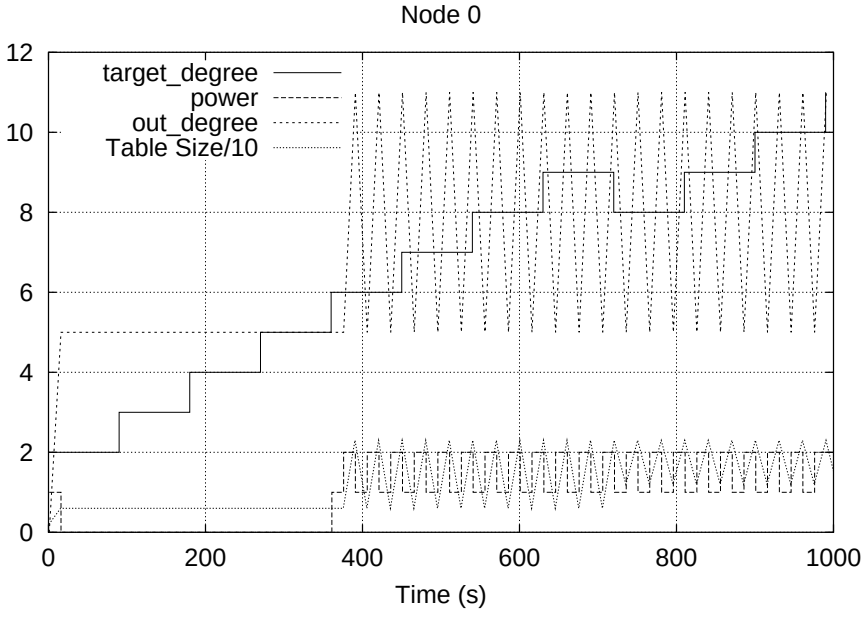



**Figure 5.12** Parameter traces for the topology of Figure 5.10.

## 5.6 Concluding Remarks

There is always a tendency, and in fact a need, to optimize performance in any system. This generally creates a tension between performance and architecture. In the case of wireless networks, we currently see this tension manifesting itself in the current interest in cross-layer design.

When engaging in cross-layer design, it may be useful to note some adverse possibilities and to exercise appropriate caution. Architecture is important for proliferation of technology, and at a time when wireless networking is rapidly becoming popular, its importance needs to be kept in mind. Unbridled cross-layer design can lead to a spaghetti design, and can stifle further innovations since the number of new interactions introduced can be large. Also, such design can stifle proliferation since every update may require a complete redesign and replacement. Moreover, cross-layer design creates interactions, some intended, others unintended. Dependency relations may need to be examined, and time scale separation may need to be enforced. The consequences of all such interactions need to be well understood, and theorems establishing stability may be needed. Proposers of cross-layer design must therefore consider the totality of the design, including the interactions with other layers, and also what other potential suggestions might be barred because they would interact with the particular proposal being made. They must also consider the architectural long term value of the suggestion. Cross-layer design proposals must therefore be holistic rather than fragmenting and it is important to adopt a cautionary approach to cross-layer design.

# CHAPTER 6

# ARCHITECTURE AND APPLICATIONS

This chapter addresses the important issue of the architecture of the network stack. Motivated by some drawbacks of the current TCP/IP stack, we propose a generalized network stack, and illustrate how it naturally and cleanly solves many problems in modern wireless networking systems. In the process we identify the need for certain services to enable the smooth operation of ad hoc networks and develop solutions to meet those needs.

Applications are driven by complex socio-economic factors which few can even claim to understand. As engineers/scientists our job is to enable technologies and make them accessible to enthusiastic users. Ad hoc networks have just about reached a stage when strong community interest has begun to develop. We will describe services that are necessary to make configuration and operation of an ad hoc network totally automatic. We have developed prototype implementations for these services, and are at the stage where all it takes to set up an ad hoc network is to plug in the wireless card. Traditional networking applications then work seamlessly.

Before describing our services we re-examine the architectural design of the current network stack and see how suitable it is for ad hoc networks.

## 6.1 Layering and Protocols: A Re-exploration

Layering is a form of abstraction particularly suited to network systems. A higher layer provides a more abstract and a more complex service while hiding the details of the lower layers. Layers are composed of objects called *protocols* that confirm to a service interface and

often a standardized algorithm. Each layer has an address associated with it. As a packet traverses down the stack each layer prefixes its headers, which enable communication with the corresponding layer at the peer (the target of the communication) node. For this, a layer needs to know the address of the corresponding layer at the peer node. This is enabled by an inter-layer address resolution service, which sits between every two layers of the protocol stack, say at every 0.5 layer as seen in Figure 6.1. The address resolution service is also accompanied by a duplicate address detection and correction service. A buffering service along with the address change service helps in providing seamless address change. The address change service also includes an auto-configuration service to initialize the address whenever possible. These services integrate the stack while keeping the layers from interacting directly. Note that these services could be implemented by providing a simple table or cache, which could be maintained at appropriate timescales by daemon programs running in user space.

### 6.1.1 Generalized network stack

As an example, let us consider the application of these principles to the popular TCP/IP Internet stack. Through this process we describe an architecturally clean "generalized" network stack.

The bottommost layer is the physical layer which is special in the sense that it has no notion of an address for itself. It is only concerned with putting bits on the medium. The data link layer provides a direct link between two or more nodes. To provide this service it may need to provide framing of bits into frames, error detection, reliable delivery and media access control.

Above the data link layer is the network layer which provides the essential service of routing packets between far away nodes. The layer 2.5 protocol which provides a mapping between network layer addresses and link layer addresses corresponds to ARP in the Internet. The duplicate address detection service at layer 2 is provided by the FCC which ensures that each manufactured network card has a unique hardware address.

Above the network layer is the transport layer which provides the abstraction of a secure reliable pipe between any two nodes in the network. It also provides a demultiplexing service for applications by providing the abstraction of ports. Let us identify the transport layer by

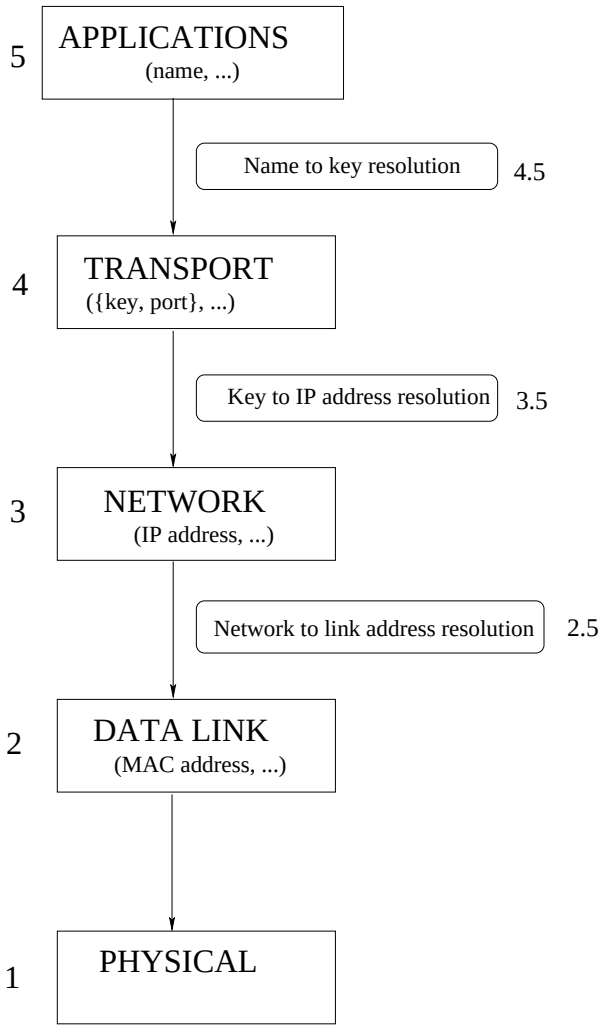



**Figure 6.1** Generalized network stack.

an address tuple of the form `(key, port)`. Above the transport layer is the application layer, which is identified by a name, and which is the only user-visible identity of a node. The `key` part in the transport layer address is really a public key which is used by the transport layer to initiate connection to a remote node. The name to key resolution service sits at layer 4.5 and provides a mapping between node names and their public keys. In the Internet this could come from trusted servers like VeriSign. In ad hoc networks there is no such trusted source, and the users probably need to rely on information extraneous to the network. However once the public keys have been reliably distributed, duplicate address detection service at this layer could be provided by a challenge/response authentication algorithm.

The key to network address resolution service sits at layer 3.5 and is the analogue of the domain name service in the Internet. There is no duplicate address detection service at layer 3.5 in the Internet and system administrators typically analyze raw data to detect IP address spoofing. The duplicate name detection service at layer 5.5 is again centralized and provided by the Internet Assigned Numbers Authority (IANA). IANA also provides part of the application name to port resolution service at layer 4.5 for the "port" part of the transport layer address. This is typically stored in the /etc/services file.

The transport layer in the Internet does not have an address of its own, rather it reuses the network layer addresses to identify a connection. Thus, a TCP connection is not immune to a change in the local IP address, which is possible in ad hoc networks due to, say, duplicate address detection. Even connectionless UDP sockets have to be reinitialized and rebound upon IP address changes. Exposing the IP addresses all the way to the application space leads to a whole set of problems created by IP address changes.

### 6.1.2 Advantages of the generalized stack

The generalized stack described above can provide a clean solution to several problems. We now elaborate on this.

TCP is unaffected by local IP address changes as the address resolution service at layer 3.5 just changes the local key to IP address mapping. Thus efforts like Rocks/Racks [118], which provide a reliable sockets library as a wrap-around to the basic socket library, become unnecessary. The key to IP address resolution service can be designed to detect changes in remote IP address, and with the help of a buffering service, can keep the higher layers oblivious to IP address changes. Mobile IP can be viewed as a mechanism which provides this service.

Security is provided as a service in the new stack at appropriate places. The scheme is robust to name spoofing as the duplicate address detection and correction service can determine the correct mapping using a challenge/response algorithm. As long as the name to key resolution service can get at least one key for every name, the recipient's identity will not be mistaken. Encryption is provided as a transport layer functionality since the key is a transport layer address. Once the connection has been established, the two ends can also switch to a symmetric

key to improve the efficiency of encryption. Thus security could be uniformly provided to all applications.

### 6.1.3 Further generalizations

Further generalizations of the above protocol stack provides many more interesting possibilities. We allow each layer to have multiple addresses. The address resolution service at each layer is then potentially a many to many mapping. Several schemes are feasible in this general architecture.

A single IP address could be associated with more then one interface. Such a configuration could be useful in seamlessly providing extra bandwidth when a node has multiple wireless cards, each in a different frequency band or using a different physical layer standard.

Conversely, a single wireless card could have multiple IP addresses associated with it. This could enable a single card to be on multiple networks (such as an ad hoc network and an infrastructure network) at virtually the same time, by switching between the two fast enough. The buffering service at layer 2.5 would make the higher layers seamless to this change. The hardware on the card may limit the performance of such a scheme but the suggested architecture provides a neat way to implement such a functionality. It could be quite useful for implementing hybrid networks or mesh networks.

A single wireless card could switch between several channels too. If the hardware allows the setting up of a different MAC address for each channel, then mapping them to a single IP address leads to interesting ways of increasing spectrum use and reducing interference in the neighborhood.

TCP connections over multiple addresses are also easily possible since the key to IP address to MAC address mapping allows it quite naturally. However it would not be a good idea for TCP to make presumptions about lower layers and consider packet reordering as an indication of loss. This is certainly a cross layer hack incompatible with the spirit of the general architecture.

## 6.2 Services for Ad Hoc Networks

Ad hoc networks should ideally confirm to the above generalized architecture to allow for all the interesting possibilities. However modifying current implementation in an operating system to the generalized architecture is beyond our current resources. Instead we[1] implement services and attempt to retrofit them into the current architecture.

### 6.2.1 ACDC: Address configuration daemon

This is a layer 3.5 service for address autoconfiguration and duplicate address detection and correction. The algorithm is simple. It takes advantage of the fact that MAC addresses are unique, and obtains the IP address by hashing based on MAC address. However, it also does not completely rule out duplicate IP addresses since IPv4 addresses are not long enough, and networks could merge or split. Duplicates are detected by monitoring the routing table for routes to the node's own IP address. Such entries could result from receiving updates from nodes having the same IP address. The selected random address is not used for a "safe period," when the node only listens to routing updates and builds up its routing table. If a duplicate is detected, then a new IP address is selected, the kernel forwarding table is flushed, and the routing daemon is restarted, since nothing can now be assumed about the network state.

### 6.2.2 DDNS: Dynamic distributed name service

This is a distributed and dynamic version of the domain name service. The DDNS daemon on each node unicasts all the name and IP address mappings it knows to all the destinations in its routing table. The frequency of updates is selected to be much lower than routing updates since names are not expected to change very fast. However, a new node joining the network reduces setup time by announcing its mapping without waiting for the update period. In the current implementation the discovered mappings are added to the /etc/hosts file which is transparently available to the applications using the `gethostbyname()` call. Efficiency can be improved by changing the system call itself to query the DDNS daemon. Whenever an address

[1]This is joint work with Vivek Raghunathan and Nitin Gupta.

changes, the name daemon is notified by ACDC and it stops sending updates. It also has to re-initialize the socket to be able to successfully send packets.

For routing, we have used in-house implementations of DSDV, Adaptive DSDV and AODV, which were done by Gupta [19]. AODV uses ASL to provide an on-demand route discovery service. This provides us with all the services necessary to have a functional ad hoc network. The services can be configured to start automatically when the user inserts a wireless card. A network then exists.

### 6.2.3 Applications

Once these services are running, any application can be used out of the box. All the regular file transfer and communication applications work as it is. However, these applications do not exploit the specific features of an ad hoc network. Recently, several distributed applications have been developed to work on overlay networks in the Internet. These can be used as it is on an ad hoc network running our services. Gnutella, Circle, Gale, ShareDaemon, Invisible IRC, etc. are some examples. The services developed here could also be useful for developing community wireless networks.

# CHAPTER 7

# CONCLUSIONS

This dissertation considers several problems regarding wireless ad hoc networks and makes contributions at several layers of the network stack.

It deals with the transmit power control problem in a comprehensive fashion. Power control affects the three important performance metrics of throughput, energy consumption and end-to-end delay. This work presents design principles which illustrate the effect of transmit power on these performance metrics. This work claims that using a low transmission power is good for improving a network's traffic carrying capacity, and can also be energy efficient depending on the energy consumption pattern of the hardware. The impact of transmit power on end-to-end delay depends on the network load. Low transmit power provides low delays when network load is high, while a higher power level is good when the network is lightly loaded, but the level of load in a network is in general difficult to determine. This work also proposes that the power control problem is properly situated at the network layer.

The above design principles are used in developing a series of increasingly complex protocols which achieve different optimizations. In general, it is not possible to simultaneously optimize all the performance metrics under all conditions. The COMPOW protocol selects a common power level for all the nodes in the network. The common power level chosen optimizes the energy consumption of all the nodes in the network subject to the constraint that the network is connected. The CLUSTERPOW protocol relaxes the constraint of common power by allowing a different transmit power level for every destination in the routing table. The routing functionality dictates not only the next hop for every destination but also the the power level to be used for that destination. The Tunneled CLUSTERPOW protocol provides a possibly finer

optimization than CLUSTERPOW but is complicated to implement. The MINPOW protocol provides energy optimal routes. The LOADPOW protocol attempts to provide low end-to-end delay by adjusting the power based on network load. It is a complex cross-layer routing scheme which needs modification of the IEEE 802.11 MAC protocol, and is difficult to implement.

COMPOW, CLUSTERPOW, and MINPOW have been implemented in the Linux kernel. The implementation architecture is fairly general, nonintrusive, and modular. It allows for a clean implementation of all the protocols and could be useful for other implementations with different goals. Thorough experimental evaluations of the implemented protocols on a testbed were not possible because of hardware limitations. The only cards capable of power control at the time of this work, the Cisco-Aironet 350 series cards, impose a huge latency for power change. The limitations are not due to the capability of the underlying electronics but rather due the way that the firmware has been programmed. To obtain some performance estimates, we implement the protocols in NS2 and perform simulation studies. The results demonstrate the benefits of power control using our protocols.

The contribution of this work at the routing layer is the development of an API to facilitate implementation of on-demand routing protocols. The API is also implemented in Linux as ASL, which is a user-space library. ASL allows for efficient implementation of on-demand routing protocols without kernel modifications, and has been used for implementing a few routing protocols including AODV.

At the transport layer extensive experimental investigations are carried out into TCP performance. The experiments conclude that setting a clamp for the sender congestion window, turning off the RTS-CTS handshake, turning on the TCP selective acknowledgments option, and setting the MAC retry limit to eight, can result in improved TCP performance with respect to the end-to-end throughput, route trip delay and delay jitter. Experiments are carried out for all possible combination of the above modifications on nine different network topologies of up to six hops, and are repeated fifteen times for each combination. Statistical testing using ANOVA techniques is used to demonstrate that the difference in performance due to the modifications is significant. Post hoc testing using multiple comparison procedures is then performed to determine those modifications to TCP that give the best performance among the ones tested.

The final run of experiments is preceded by several smaller experiments to understand the effect of various factors on TCP performance. The various factors that have been studied include congestion window clamp, MAC retry limit, packet sizes, suitability of routing protocols, suitability of automatic rate adaptation, effect of router queue lengths, and details about the MAC backoff algorithm. All these experiments are archived and publicly available for download. Immediate on-line processing of data is also possible through a CGI interface.

We learned several important techniques for setting up the experiments so that they could be performed manageably and repeatably. An important one was covering the wireless cards with copper tape to reduce their range, so that multihop topologies could be formed in a limited space, such as a hall. When forming topologies it is critical to ensure that links are stable with a low loss rate. The background traffic was collected using the Kismet wireless sniffer. The Linux kernel was modified to output TCP connection parameters to the *proc* file system. Finally, a set of shell scripts were written which completely automated the whole process of setting up parameters, starting traffic sources on multiple nodes, and logging all possible data at various nodes. This allowed us to run several experiments smoothly. Another set of perl scripts were written to process the logged data and generate summary statistics. These can be executed through the CGI interface too.

To automate the setting up of an ad hoc network some services are necessary in the current TCP/IP stack. These include an IP address autoconfiguration service and a distributed domain name service. We have provided prototype implementations in Linux for these services. Typical networking applications run out of the box once our services are started, which can be programmed to happen at the insertion of a wireless card.

To tie everything together, this work considers the issue of architecture as a whole. First, it presents a cautionary perspective on cross-layer design illustrating the various pitfalls. Then the TCP/IP stack is scrutinized for cross-layer hacks, which leads to the development of a generalized stack (Chapter 6). In the generalized stack each layer is represented by a tuple of addresses. Communication between layers is allowed only through a set of services, which reside at every 0.5 layer. These include the address resolution service which provides a many-to-many mapping between the addresses of the two layers between which it sits. Each layer also has an address autoconfiguration and duplicate detection and correction service. Several examples

illustrate how such a generalized stack provides a clean solution to various needs of dynamic wireless networking systems.

To conclude, we have dealt holistically with several problems in wireless ad hoc networks. We hope these contributions constitute some measure of progress with respect to the design and deployment of wireless ad hoc networks.

# APPENDIX A

# SCHEFFE'S MULTIPLE COMPARISON TEST

This appendix presents the results of Scheffe's post hoc multiple comparison range tests on the experimental data of Chapter 3. Scheffe's test groups the means into homogeneous subsets based on a given $\alpha$ level ($\alpha = 0.05$ here). The values listed in the following tables are the group means and the significance values for the within subset differences. Tables A.1, A.2, and A.3 list the homogeneous subsets for the response variables: throughput, delay, and delay jitter, respectively.

Table A.1: Homogenous subsets for throughput.

| Treatment | Subsets for $\alpha = .05$ | | | | |
|---|---|---|---|---|---|
| 1hop topology | 1 | 2 | 3 | 4 | 5 |
| 6 | 3.17923 | | | | |
| 2 | 3.19923 | | | | |
| 1 | | 3.33308 | | | |
| 5 | | 3.33308 | | | |
| 0 | | | 4.11615 | | |
| 4 | | | | 4.29154 | |
| 3 | | | | | 4.53538 |
| 7 | | | | | 4.64385 |
| Sig. | 0.999 | 1.000 | 1.000 | 1.000 | 0.054 |
| 2 hop topology | 1 | 2 | 3 | 4 | 5 |
| 2 | 1.91846 | | | | |
| 6 | 1.92000 | | | | |
| 0 | | 2.02692 | | | |
| 1 | | 2.06077 | 2.06077 | | |
| 5 | | 2.06462 | 2.06462 | | |
| 4 | | | 2.07692 | | |

*continued on next page*

*Table A.1 continued*

| | | | | | |
|---|---|---|---|---|---|
| 3 | | | | 2.19000 | |
| 7 | | | | | 2.26462 |
| Sig. | 1.000 | 0.098 | 0.939 | 1.000 | 1.000 |

| 3 hop topology | 1 | 2 | 3 | | |
|---|---|---|---|---|---|
| 0 | 1.28462 | | | | |
| 6 | 1.32231 | 1.32231 | | | |
| 4 | 1.34385 | 1.34385 | | | |
| 2 | 1.35769 | 1.35769 | 1.35769 | | |
| 3 | | 1.40923 | 1.40923 | | |
| 5 | | | 1.45000 | | |
| 1 | | | 1.45308 | | |
| 7 | | | 1.45462 | | |
| Sig. | 0.358 | 0.150 | 0.067 | | |

| 4 hop topology | 1 | 2 | | | |
|---|---|---|---|---|---|
| 0 | 1.01769 | | | | |
| 2 | 1.01846 | | | | |
| 4 | 1.02231 | | | | |
| 6 | 1.02692 | | | | |
| 3 | 1.08077 | 1.08077 | | | |
| 5 | | 1.10692 | | | |
| 7 | | 1.11308 | | | |
| 1 | | 1.12000 | | | |
| Sig. | 0.060 | 0.604 | | | |

| 5 hop topology | 1 | 2 | 3 | 4 | |
|---|---|---|---|---|---|
| 4 | 0.77592 | | | | |
| 0 | 0.78023 | | | | |
| 2 | 0.79462 | 0.79462 | | | |
| 6 | 0.80146 | 0.80146 | | | |
| 3 | | 0.84346 | 0.84346 | | |
| 1 | | | 0.86438 | 0.86438 | |
| 7 | | | 0.88523 | 0.88523 | |
| 5 | | | | 0.89692 | |
| Sig. | 0.838 | 0.096 | 0.251 | 0.590 | |

| 6 hop topology | 1 | 2 | 3 | | |
|---|---|---|---|---|---|
| 6 | 0.51962 | | | | |
| 2 | 0.52015 | | | | |
| 3 | 0.52523 | 0.52523 | | | |
| 0 | 0.52731 | 0.52731 | | | |
| 4 | 0.53400 | 0.53400 | 0.53400 | | |
| 5 | 0.55462 | 0.55462 | 0.55462 | | |
| 7 | | 0.56869 | 0.56869 | | |

*continued on next page*

| *Table A.1 continued* | | | | | |
|---|---|---|---|---|---|
| 1 | | | 0.57677 | | |
| Sig. | 0.361 | 0.114 | 0.127 | | |
| 2by2 hop topology | 1 | 2 | 3 | | |
| 6 | 1.74909 | | | | |
| 2 | 1.77727 | | | | |
| 5 | 1.89727 | 1.89727 | | | |
| 0 | 1.91455 | 1.91455 | | | |
| 1 | 1.93545 | 1.93545 | 1.93545 | | |
| 4 | | 1.97455 | 1.97455 | | |
| 3 | | 2.05273 | 2.05273 | | |
| 7 | | | 2.11364 | | |
| Sig. | 0.071 | 0.233 | 0.101 | | |
| 3by2 hop topology | 1 | 2 | 3 | 4 | 5 |
| 6 | 1.46923 | | | | |
| 2 | 1.47923 | | | | |
| 5 | | 1.59462 | | | |
| 1 | | 1.60231 | | | |
| 4 | | | 1.72385 | | |
| 0 | | | 1.77923 | 1.77923 | |
| 3 | | | | 1.84385 | 1.84385 |
| 7 | | | | | 1.94000 |
| Sig. | 1.000 | 1.000 | 0.821 | 0.671 | 0.159 |
| 4by2 hop topology | 1 | | | | |
| 6 | 1.03215 | | | | |
| 2 | 1.05877 | | | | |
| 0 | 1.07969 | | | | |
| 4 | 1.09985 | | | | |
| 1 | 1.11623 | | | | |
| 5 | 1.13892 | | | | |
| 7 | 1.21000 | | | | |
| 3 | 1.22531 | | | | |
| Sig. | 0.106 | | | | |

Table A.2: Homogenous subsets for RTT average.

| Treatment | Subsets for $\alpha = .05$ | | | | | |
|---|---|---|---|---|---|---|
| 1 hop topology | 1 | 2 | 3 | | | |
| 1 | 8.638 | | | | | |
| 5 | 8.638 | | | | | |
| 2 | 9.062 | | | | | |

*continued on next page*

*Table A.2 continued*

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 9.131 | | | | | |
| 3 | | 39.315 | | | | |
| 7 | | 40.277 | | | | |
| 0 | | | 68.908 | | | |
| 4 | | | 70.177 | | | |
| Sig. | 1.000 | 1.000 | 1.000 | | | |
| 2 hop topology | 1 | 2 | 3 | | | |
| 5 | 17.054 | | | | | |
| 1 | 17.131 | | | | | |
| 6 | 18.423 | | | | | |
| 2 | 18.485 | | | | | |
| 7 | | 590.346 | | | | |
| 4 | | 655.862 | 655.862 | | | |
| 3 | | 687.754 | 687.754 | | | |
| 0 | | | 750.577 | | | |
| Sig. | 1.000 | .214 | .247 | | | |
| 3 hop topology | 1 | 2 | | | | |
| 1 | 49.200 | | | | | |
| 5 | 49.369 | | | | | |
| 2 | 53.331 | | | | | |
| 6 | 54.954 | | | | | |
| 4 | | 1074.469 | | | | |
| 7 | | 1077.454 | | | | |
| 3 | | 1148.792 | | | | |
| 0 | | 1237.062 | | | | |
| Sig. | 1.000 | .478 | | | | |
| 4hop topology | 1 | 2 | | | | |
| 1 | 88.615 | | | | | |
| 5 | 89.862 | | | | | |
| 6 | 96.846 | | | | | |
| 2 | 97.100 | | | | | |
| 7 | | 759.985 | | | | |
| 0 | | 838.577 | | | | |
| 4 | | 863.054 | | | | |
| 3 | | 919.138 | | | | |
| Sig. | 1.000 | .981 | | | | |
| 5 hop topology | 1 | 2 | 3 | 4 | 5 | 6 |
| 5 | 132.892 | | | | | |
| 1 | 133.769 | 133.769 | | | | |
| 2 | | 143.100 | 143.100 | | | |
| 6 | | | 145.515 | | | |

*continued on next page*

*Table A.2 continued*

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | | | | 170.469 | | |
| 7 | | | | 174.215 | 174.215 | |
| 0 | | | | | 180.892 | 180.892 |
| 4 | | | | | | 186.023 |
| Sig. | 1.000 | .086 | .996 | .953 | .473 | .786 |
| 6 hop topology | 1 | 2 | | | | |
| 5 | 182.623 | | | | | |
| 1 | 183.400 | | | | | |
| 6 | 194.600 | | | | | |
| 2 | 195.285 | | | | | |
| 3 | 270.415 | 270.415 | | | | |
| 7 | | 291.315 | | | | |
| 0 | | 304.069 | | | | |
| 4 | | 333.423 | | | | |
| Sig. | .072 | .439 | | | | |
| 2by2 hops topology | 1 | 2 | | | | |
| 1 | 43.418 | | | | | |
| 5 | 45.095 | | | | | |
| 2 | 48.805 | | | | | |
| 6 | 50.264 | | | | | |
| 3 | | 702.650 | | | | |
| 7 | | 716.577 | | | | |
| 0 | | 742.964 | | | | |
| 4 | | 790.255 | | | | |
| Sig. | 1.000 | .072 | | | | |
| 3by2 hops topology | 1 | 2 | | | | |
| 5 | 100.423 | | | | | |
| 1 | 100.654 | | | | | |
| 2 | 109.269 | | | | | |
| 6 | 110.477 | | | | | |
| 3 | | 513.277 | | | | |
| 4 | | 519.515 | | | | |
| 7 | | 520.400 | | | | |
| 0 | | 569.377 | | | | |
| Sig. | 1.000 | .802 | | | | |
| 4by2 hops topology | 1 | 2 | | | | |
| 5 | 152.477 | | | | | |
| 1 | 154.477 | | | | | |
| 2 | 161.631 | | | | | |
| 6 | 169.338 | | | | | |
| 3 | | 340.346 | | | | |

*continued on next page*

| *Table A.2 continued* | | | | | | |
|---|---|---|---|---|---|---|
| 7 | | 342.192 | | | | |
| 0 | | 350.369 | | | | |
| 4 | | 377.038 | | | | |
| Sig. | .999 | .899 | | | | |

Table A.3: Homogenous subsets for St. deviation of RTT.

| Treatment | Subsets for $\alpha = .05$ | | |
|---|---|---|---|
| 1 hop topology | 1 | 2 | 3 |
| 2 | 3.738 | | |
| 6 | 3.838 | | |
| 1 | 3.877 | | |
| 5 | 3.908 | | |
| 3 | | 34.531 | |
| 7 | | 36.262 | |
| 4 | | | 74.369 |
| 0 | | | 78.285 |
| Sig. | 1.000 | 1.000 | 0.999 |
| 2 hop topology | 1 | 2 | 3 |
| 5 | 5.392 | | |
| 1 | 5.515 | | |
| 6 | 5.769 | | |
| 2 | 6.015 | | |
| 7 | | 160.515 | |
| 3 | | 182.538 | 182.538 |
| 4 | | | 203.046 |
| 0 | | | 205.623 |
| Sig. | 1.000 | 0.549 | 0.485 |
| 3 hop topology | 1 | 2 | 3 |
| 5 | 15.415 | | |
| 1 | 16.038 | | |
| 2 | 16.677 | | |
| 6 | 17.831 | | |
| 7 | | 249.508 | |
| 3 | | 282.069 | 282.069 |
| 4 | | 286.208 | 286.208 |
| 0 | | | 307.423 |
| Sig. | 1.000 | 0.481 | 0.869 |
| 4 hop topology | 1 | 2 | 3 |
| 1 | 22.869 | | |

*continued on next page*

*Table A.3 continued*

| | | | |
|---|---|---|---|
| 5 | 23.746 | | |
| 6 | 24.854 | | |
| 2 | 25.231 | | |
| 0 | | 248.915 | |
| 7 | | 325.969 | 325.969 |
| 4 | | 332.177 | 332.177 |
| 3 | | | 378.792 |
| Sig. | 1.000 | 0.324 | 0.852 |

| 5 hop topology | 1 | 2 | |
|---|---|---|---|
| 5 | 40.254 | | |
| 1 | 41.931 | | |
| 6 | 43.162 | | |
| 2 | 47.138 | | |
| 0 | | 69.638 | |
| 7 | | 70.362 | |
| 4 | | 70.815 | |
| 3 | | 71.523 | |
| Sig. | 0.914 | 1.000 | |

| 6 hop topology | 1 | 2 | |
|---|---|---|---|
| 5 | 74.769 | | |
| 1 | 77.485 | | |
| 6 | 77.815 | | |
| 2 | 87.231 | | |
| 7 | 189.462 | 189.462 | |
| 0 | 191.231 | 191.231 | |
| 3 | 191.862 | 191.862 | |
| 4 | | 246.908 | |
| Sig. | 0.263 | 0.947 | |

| 2by2 hop topology | 1 | 2 | |
|---|---|---|---|
| 1 | 21.432 | | |
| 5 | 22.118 | | |
| 2 | 23.418 | | |
| 6 | 27.464 | | |
| 4 | | 240.818 | |
| 7 | | 247.241 | |
| 3 | | 252.277 | |
| 0 | | 277.041 | |
| Sig. | 1.000 | 0.083 | |

| 3by2 hop topology | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 40.469 | | |
| 5 | 40.708 | | |

*continued on next page*

*Table A.3 continued*

| | | | |
|---|---|---|---|
| 6 | 41.385 | | |
| 2 | 41.431 | | |
| 4 | | 220.323 | |
| 3 | | 245.846 | 245.846 |
| 7 | | 256.669 | 256.669 |
| 0 | | | 274.538 |
| Sig. | 1.000 | 0.146 | 0.440 |

| 4by2 hop topology | 1 | 2 | |
|---|---|---|---|
| 5 | 57.300 | | |
| 1 | 60.277 | | |
| 6 | 61.492 | | |
| 2 | 62.254 | | |
| 0 | | 241.238 | |
| 4 | | 241.831 | |
| 7 | | 245.877 | |
| 3 | | 251.046 | |
| Sig. | 1.000 | 1.000 | |

# REFERENCES


[1] P. Karn, "MACA – A new channel access method for packet radio," in *Proceedings of Computer Networking Conference*, 1990.

[2] V. Bharghavan, A. J. Demers, S. Shenker, and L. Zhang, "MACAW: A media access protocol for wireless LANs," in *SIGCOMM*, 1994, pp. 212–225.

[3] IEEE 802 LAN/MAN Standards Committee, "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," IEEE Standard 802.11, 1999 edition, 1999.

[4] N. Bambos, S. C. Chen, and G. J. Pottie, "Channel access algorithms with active link protection for wireless communication networks with power control," *IEEE/ACM Transactions on Networking (TON)*, vol. 8, no. 5, pp. 583–597, 2000.

[5] S. Ulukus and R. Yates, "Stochastic power control for cellular radio systems," *IEEE Transactions Communications*, vol. 46, no. 6, pp. 784–798, 1998.

[6] S. Wu, Y. Tseng, and J. Sheu, "Intelligent medium access for mobile ad hoc networks with busy tones and power control," *IEEE Journal on Selected Area in Communications*, vol. 18, no. 9, pp. 1647–57, 2000.

[7] J. P. Monks, V. Bhargavan, and W.-M. Hwu, "A power controlled multiple access protocol for wireless packet networks," in *Proceedings of IEEE INFOCOM*, 2001, pp. 219–228.

[8] R. Ramanathan and R. Rosales-Hain, "Topology control of multihop wireless networks using transmit power adjustment," in *Proceedings of IEEE INFOCOM*, 2000, pp. 404–413.

[9] R. Wattenhofer, L. Li, P. Bahl, and Y.-M. Wang, "Distributed topology control for power efficient operation in multihop wireless ad hoc networks," in *Proceedings of IEEE INFOCOM*, 2001, pp. 1388–1397.

[10] Q. Li, J. Aslam, and D. Rus, "Online power-aware routing in wireless ad-hoc networks," in *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking*, July 2001, pp. 97–107.

[11] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *ACM Wireless Networks*, vol. 1, no. 4, pp. 469–481, 1995.

[12] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *15th International Conference on Distributed Computing Systems*, 1995.

[13] S. Biaz and N. H. Vaidya, "Discriminating congestion losses from wireless losses using inter-arrival times at the receiver," in *IEEE Symposium ASSET*, 1999, pp. 10–18.

[14] J. Liu and S. Singh, "ATCP: TCP for mobile ad hoc networks," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 7, pp. 1300–1315, July 2001.

[15] Z. Fu, B. Greenstein, X. Meng, and S. Lu, "Design and implementation of a TCP-friendly transport protocol for ad hoc wireless networks," in *IEEE International Conference on Network Protocols'02*, 2002.

[16] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A feedback-based scheme for improving TCP performance in ad hoc wireless networks," *IEEE Personal Communications Magazine*, vol. 8, no. 1, pp. 34–39, Feb. 2001.

[17] F. Wang and Y. Zhang, "Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response," in *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2002, pp. 217–225.

[18] L. Breslau, et al., "Advances in network simulation," *IEEE Computer*, vol. 33, no. 5, pp. 59–67, May 2000.

[19] B. Gupta, "Design, implementation and testing of routing protocols for mobile ad-hoc networks," M.S. thesis, University of Illinois at Urbana-Champaign, 2002.

[20] P. Gupta and P. R. Kumar, "Critical power for asymptotic connectivity in wireless network," in *Stochastic Analysis, Control, Optimization and Applications: A Volume in Honor of W.H. Fleming*, W. M. McEneany, G. Yin, and Q. Zhan, Eds. Boston: Birkhauser, 1998, pp. 547–566.

[21] F. Xue and P. R. Kumar, "The number of neighbors needed for connectivity of wireless networks," *Wireless Networks*, vol. 10, no. 2, pp. 169–181, 2004.

[22] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Transactions on Information Theory*, vol. IT-46, pp. 388–404, 2000.

[23] C. E. Perkins, E. M. Royer, and S. Das, "Ad hoc on demand distance vector routing," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 1999, pp. 90–100.

[24] D. B. Johnson and D. A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*, T. Imielinski and H. Korth, Eds. Boston: Kluwer Academic Publishers, 1996.

[25] V. Kawadia and P. R. Kumar, "A cautionary perspective on cross-layer design," *IEEE Wireless Communication Magazine*, 2004, to be published.

[26] V. Kawadia and P. R. Kumar, "Principles and protocols for power control in ad hoc networks," *IEEE Journal on Selected Areas in Communications: Special Issue on Ad Hoc Networks*, 2004, to be published.

[27] S. Narayanaswamy, V. Kawadia, R. S. Sreenivas, and P. R. Kumar, "Power control in ad-hoc networks: Theory, architecture, algorithm and implementation of the COMPOW protocol," in *European Wireless Conference*, 2002, pp. 156–162.

[28] V. Kawadia and P. R. Kumar, "Power control and clustering in ad-hoc networks," in *Proceedings of IEEE INFOCOM*, no. 1, 2003, pp. 459–469.

[29] T. J. Shepard, "A channel access scheme for large dense packet radio networks," in *Proceedings of ACM SIGCOMM*, 1996, pp. 219–230.

[30] "Data sheet: Cisco aironet 350 series client adapters," 1992, http://www.cisco.com/en/US/products/hw/wireless/ps4555/ps448/index.html.

[31] E. Shih, P. Bahl, and M. Sinclair, "Wake on wireless: An event driven energy saving strategy for battery operated devices," in *Proceedings of ACM MOBICOM*, 2002, pp. 160–171.

[32] R. Zheng, J. Hou, and L. Sha, "Asynchronous wakeup for ad hoc networks," in *The Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing MobiHoc*, 2003, pp. 35–45.

[33] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks," in *Proceedings of ACM MOBICOM*, 2001, pp. 85–96.

[34] Y. Xu, J. S. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," in *Proceedings of ACM MOBICOM*, 2001, pp. 70–84.

[35] C. E. Perkins and P. R. Bhagwat, "Highly dynamic destination-sequenced distance vector routing (DSDV) for mobile computers," in *Proceedings of ACM SIGCOMM*, 1994, pp. 234–244.

[36] V. Kawadia, Y. Zhang, and B. Gupta, "System services for ad hoc routing: Architecture, implementation and experiences," in *ACM MOBISYS*, 2003.

[37] V. Kawadia and P. R. Kumar, "The wireless software page," 2004, http://decision.csl.uiuc.edu/∼wireless/.

[38] S. Singh, M. Woo, and C. S. Raghavendra, "Power aware routing in mobile ad hoc networks," in *Proceedings of ACM MOBICOM*, 1998, pp. 181–190.

[39] M. W. Subbarao, "Dynamic power-conscious routing for manets: An initial approach," in *IEEE Vehicular Technology Conference*, 1999, pp. 1232–1237.

[40] R. Dube, C. D. Rais, K.-Y. Wang, and S. K. Tripathi, "Signal stability based adaptive routing (SSA) for ad-hoc mobile networks," in *IEEE Personal Communications*, 1997, pp. 36–45.

[41] E. L. Lloyd, R. Liu, M. Marathe, R. Ramanathan, and S. S. Ravi, "Algorithmic aspects of topology control problems for ad hoc networks," in *The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Laussane, Switzerland, 2002, pp. 123–134.

[42] T. A. Elbatt, S. V. Krishnamurthy, D. Connors, and S. Dao, "Power management for throughput enhancement in wireless ad-hoc networks," in *IEEE International Conference on Communications*, 2000, pp. 1506–1513.

[43] A. Michail and A. Ephremides, "Energy-efficient routing for connection-oriented traffic in wireless ad-hoc networks," *Mobile Networks and Applications*, vol. 8, no. 5, pp. 517–533, 2003.

[44] G. Zussman and A. Segall, "Energy efficient routing in ad hoc disaster recovery networks," in *Proceedings of IEEE INFOCOM*, 2003, pp. 682–691.

[45] J. Wieselthier, G. Nguyen, and A. Ephremides, "Energy-efficient broadcast and multicast trees in wireless networks," *Mobile Networks and Applications (MONET)*, vol. 7, no. 6, pp. 481–492, December 2002.

[46] M. Cagalj and J.-P. Hubaux, "Minimum-energy broadcast in all-wireless networks: Np-completeness and distribution issues," in *ACM MOBICOM*, 2002, pp. 172–182.

[47] J. Wieselthier, G. Nguyen, and A. Ephremides, "Energy-aware wireless networking with directional antennas: The case of session-based broadcasting and multicasting," *IEEE Transactions on Mobile Computing*, vol. 1, no. 3, pp. 176–191, July-Sept 2002.

[48] J. P. Monks, V. Bhargavan, and W.-M. Hwu, "A power controlled multiple access protocol for wireless packet networks," in *Proceedings of IEEE INFOCOM*, 2001, pp. 219–228.

[49] S. Singh and C. S. Raghavendra, "Power efficient MAC protocol for multihop radio networks," in *The Ninth IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 1998, pp. 153–157.

[50] E.-S. Jung and N. H. Vaidya, "A power control MAC protocol for ad-hoc networks," in *ACM MOBICOM*, 2002, pp. 36–47.

[51] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh, "Power-saving protocols for IEEE 802.11-based multi-hop ad hoc networks," in *IEEE INFOCOM 2002 – The Conference on Computer Communications*, 2000, pp. 200–209.

[52] S.-J. Park and R. Sivakumar, "Load sensitive transmission power control in wireless ad-hoc networks," in *Proceedings of IEEE GLOBECOM*, 2002, pp. 42–46.

[53] R. E. Ludwig, "Eliminating inefficient cross-layer interactions in wireless networking," Ph.D. dissertation, Aachen University of Technology, Germany, 2000.

[54] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions in Networking*, vol. 5, no. 6, pp. 756–769, Dec 1997.

[55] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *Proceedings of ACM MOBICOM*, Aug. 1999, pp. 275–288.

[56] G. Holland and N. Vaidya, "Impact of routing and link layers on TCP performance in mobile ad hoc networks," in *IEEE Wireless Communications and Networking Conference*, 1999, pp. 1323–1327.

[57] V. Anantharaman, S.-J. Park, K. Sundaresan, and R. Sivakumar, "TCP performance over mobile ad-hoc networks: A quantitative study," *Wireless Communications and Mobile Computing Journal, Special Issue on Performance Evaluation of Wireless Network*, 2003.

[58] D. Kim, C.-K. Toh, and Y. Choi, "TCP-BuS: Improving TCP performance in wireless ad hoc networks," in *IEEE International Conference on Communications*, June 2000, pp. 1707–1713.

[59] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on TCP throughput and loss," in *Proceedings of IEEE INFOCOM*, 2003, pp. 1744–1753.

[60] K. Chen, Y. Xue, S. H. Shah, and K. Nahrstedt, "Understanding bandwidth-delay product in mobile ad hoc networks," *Elsevier Computer Commnications Journal*, vol. 27, no. 10, pp. 923–934, 2004.

[61] G. Anastasi, E. Borgia, M. Conti, and E. Gregor, "IEEE 802.11 ad hoc networks: Performance measurement," in *Workshop on Mobile and Wireless Networks in Conjunction with ICDCS*, 2003, pp. 758–763.

[62] D. Dhoutaut and I. G. Lassou, "Experiments with 802.11b in ad hoc configuration," in *IEEE International Symposium on Personal Indoor and Mobile Radio Communication*, 2003, pp. 1618–1622.

[63] M. Kershaw, "Kismet," 2003, http://www.kismetwireless.net/.

[64] E. Bartel, "Nttcp: new ttcp program," 1997, http://www.leo.org/∼elmar/nttcp/.

[65] B. Hubert, "Linux advanced routing and traffic control," 2002, http://www.lartc.org/.

[66] M. Rio, M. Goutelle, T. Kelly, R. Highes-Jones, J.-P. Martin-Flatin, and Y.-T. Li, "A map of the networking code in Linux kernel 2.4.20," 2003, http://datatag.web.cern.ch/datatag/papers/tr-datatag-2004-1.pdf.

[67] S. Ostermann, "Tcptrace: Official homepage," 2000, http://www.tcptrace.org/.

[68] P. Sarolahti and A. Kuznetsov, "Congestion control in Linux TCP," in *USENIX Annual Technical Conference*, 2002, pp. 49–62.

[69] O. Andreasson, "IPsysctl tutorial 1.0.4," 2002, http://ipsysctl-tutorial.frozentux.net/.

[70] A. Cuff, "Wireless security tools," 1999, http://www.networkintrusion.co.uk/wireless.htm.

[71] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "NLANR/DAST: Iperf 1.7.0 – the TCP/UDP bandwidth measurement tool," 2003, http://dast.nlanr.net/Projects/Iperf/.

[72] F. P. Kelly, *Reversibility and Stochastic Networks*. Cambridge: John Wiley and Sons Ltd., 1979, pp. 34–48.

[73] H. Jin, J. Ou, and P. R. Kumar, "The throughput of irreducible closed Markovian queueing networks: Functional bounds, asymptotic loss, efficiency, and the Harrison-Wein conjectures," *Mathematics of Operations Research*, vol. 22, no. 4, pp. 886–920, 1997.

[74] N. Gupta and P. R. Kumar, "A performance analysis of the IEEE 802.11 wireless LAN medium access control," *Communications in Information and Systems*, to be published.

[75] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proceedings of ACM MOBICOM*, 2003, pp. 134–146.

[76] A. Kamerman and L. Monteban, "WaveLAN II: A high-performance wireless LAN for the unlicensed band," *Bell Labs Technical Journal*, vol. 1, pp. 118–133, 1997.

[77] B. Awerbuch, D. Holmer, and H. Rubens, "High throughput route selection in multi-rate ad hoc wireless networks," in *Wireless On-Demand Network Systems*, 2004.

[78] S. Kunniyur and R. Srikant, "An adaptive virtual queue (AVQ) algorithm for active queue management," *IEEE/ACM Transaction on Networking*, vol. 12, no. 2, pp. 286–299, 2004.

[79] "PROPHET statguide," 1996, http://www.basic.nwu.edu/statguidefiles/list.html.

[80] M. J. Roberts and R. Russo, *A Students's Guide to Analysis of Variance*. London: Routledge, 1999.

[81] R. S. Bogartz, *An Introduction to the Analysis of Variance*. Westport, Connecticut: Praeger, 1994.

[82] "PROPHET statguide: One-way analysis of variance (ANOVA)," 1996, http://www.basic.nwu.edu/statguidefiles/oneway_anova.html.

[83] "SPSS 12.0: Base system reference manual," 2004, http://www.spss.com.

[84] Y. Hochberg and A. C. Tamhane, *Multiple Comparison Procedures*. New York: John Wiley & Sons, 1987.

[85] L. L. Peterson and B. S. Davie, *Computer Networks*, 2nd ed. San Diego: Morgan Kaufmann Publishers, 2000.

[86] E. S. Raymond, *The Art of Unix Programming*. Boston: Addison-Wesley, 2003.

[87] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *Proceedings of ACM SIGCOMM'94*, London, U.K., Sept. 1994.

[88] Y.-C. Hu and D. B. Johnson, "Implicit source routes for on-demand ad hoc network routing," in *Proceedings of ACM MOBIHOC*, Long Beach, California, Oct. 2001, pp. 1–10.

[89] R. Russell, "Netfilter/Iptables homepage," 1999, http://www.netfilter.org.

[90] E. M. Royer and C. E. Perkins, "An implemenatation study of the aodv routing protocol," in *Proceedings of the IEEE Wireless Communications and Networking Conference*, 2000.

[91] H. Lundgren and E. Nordstrom, "AODV-Uppasala University," 2002, http://www.docs.uu.se/∼henrikl/aodv.

[92] C. Perkins, E. Belding-Royer, S. Das, and I. Chakeres, "AODV homepage," 2001, http://moment.cs.ucsb.edu/AODV/aodv.html.

[93] L. Klein-Berndt, "Kernel AODV," 2001, http://w3.antd.nist.gov/wctg/aodv_kernel.

[94] S. Desilva and S. Das, "Experimental evaluation of a wireless ad hoc network," in *Proceedings of the 9th International Conference on Computer Communications and Networks*, 2000, pp. 414–421.

[95] D. Maltz, J. Broch, and D. Johnson, "Experiences designing and building a multi-hop wireless ad hoc network testbed," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-99-116, 1999.

[96] D. Maltz, J. Broch, and D. Johnson, "Implementation of DSR," 1999, http://www.monarch.cs.cmu.edu/dsr-impl.html.

[97] S. Corson, S. Papademetriou, P. Papadopoulos, V. Park, and A. Qayyum, "An internet MANET encapsulation protocol (IMEP) specification," Aug 1999, iETF Draft, draft-ietf-manet-imep-spec02.txt, work in progress.

[98] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proceedings of IEEE INFOCOM*, Kobe, Japan, April 1997.

[99] A. Tornquis, "A modular framework for implementing ad hoc routing protocols," M.S. thesis, University of Colorado, 2000, http://systems.cs.colorado.edu/Networking/modular-adhoc.html.

[100] N. K. Palanisam, "Modular implementation of temporally ordered routing algorithm," M.S. thesis, University of Colorado, 2001.

[101] S. Doshi, S. Bhandare, and T. X. Brown, "An on-demand minimum energy routing protocol for a wireless ad hoc network," *Mobile Computing and Communications Review*, vol. 6, no. 2, pp. 50–66, July 2002.

[102] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[103] F. Kargl, J. Nagler, and S. Schlott, "Building a framework for manet routing protocols," 2003, http://medien.informatik.uni-ulm.de/∼frank/research/manetframework.pdf.

[104] R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. D. Kim, B. Zhou, and E. G. Sirer, "On the need for system-level support for ad hoc and sensor networks," *ACM Operating Systems Review*, vol. 36, no. 2, pp. 1–5, Apr. 2002.

[105] L.-L. Xie and P. R. Kumar, "A network information theory for wireless communication: Scaling laws and optimal operation," *IEEE Transactions on Information Theory*, vol. 50, no. 5, pp. 748–767, 2004.

[106] B. Schein and R. Gallager, "The Gaussian parallel relay network," in *IEEE International Symposium on Information Theory*, 2000, p. 22.

[107] A. W. Burks, H. H. Goldstine, and J. von Neumann, *John von Neumann Collected Works, Volume V*, A. H. Taub, Ed. New York: The Macmillan Co., 1963.

[108] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[109] C. Shannon and W. Weaver, *The Mathematical Theory of Communication*. Urbana, Illinois: University of Illinois Press, 1949.

[110] M. Franceschetti, J. Bruck, and L. Schulman, "A random walk model of wave propagation," *IEEE Transactions on Antennas and Propagation*, vol. 52, no. 5, pp. 1304–1317, May 2004.

[111] J. Zhang, "ICC panel on defining cross-layer design in wireless networking," 2003, http://www.eas.asu.edu/∼junshan/ICC03panel.html.

[112] K. J. Åström and B. Wittenmark, *Adaptive Control*. Boston: Addison-Wesley, 1995.

[113] P. R. Kumar, "A survey of some results in stochastic adaptive control," *SIAM Journal on Control and Optimization*, vol. 23, no. 3, pp. 329–380, 1985.

[114] L. Ljung, "Analysis of recursive stochastic algorithms," *IEEE Transactions on Automatic Control*, vol. AC-22, pp. 551–575, 1977.

[115] G. Holland, N. Vaidya, and P. Bahl, "A rate-adaptive MAC protocol for multi-hop wireless networks," in *Proceedings of ACM MOBICOM*, 2001, pp. 236–251.

[116] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly, "Opportunistic media sccess for multirate ad hoc networks," in *Proceedings of ACM MOBICOM*, 2002, pp. 24–35.

[117] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knightly, "OAR implementation in NS2," 2003, http://www-ece.rice.edu/networks/software/OAR/OAR.html.

[118] V. C. Zandy and B. P. Miller, "Reliable network connections," in *Proceedings of ACM MOBICOM*, 2002, pp. 95–106.

The Road goes ever on and on
Out from the door where it began.
Now far ahead the Road has gone,
Let others follow it who can!
Let them a journey new begin,
But I at last with weary feet
Will turn towards the lighted inn,
My evening-rest and sleep to meet.[1]

[1]Bilbo Baggins, after his adventure.

# VITA

Vikas Kawadia was born in 1977 in Udaipur, Rajasthan, India. He grew up in Jabalpur and Indore in the state of Madhya Pradesh. He received the degree of Bachelor of Technology in Engineering Physics from the Indian Institute of Technology Bombay, Mumbai in 1999. He received the Master of Science in Electrical Engineering degree from Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign in 2001.

Vikas is a recipient of the E. A. Reid Award from the department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. He has also received the National Talent Search scholarship in 1993, awarded by NCERT, India. His research has focused on wireless ad hoc networks and includes design, analysis, implementation, and experimentation of protocols. After completing his PhD, he will begin working as a Network Scientist at BBN Technologies in Cambridge, Massachusetts.