MULTICHANNEL WIRELESS NETWORKS: CAPACITY AND PROTOCOLS

BY

PRADEEP NARAYANASWAMY KYASANUR

B.E, Mangalore University, 2001
M.S, University of Illinois at Urbana-Champaign, 2003

DISSERTATION

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

Urbana, Illinois

ABSTRACT

Recent years have seen significant interest in using the multihop wireless networking paradigm for building mesh networks, ad hoc networks, and sensor networks. A key challenge in multihop wireless networks is to provision for sufficient network capacity to meet user requirements. Several approaches have been proposed to improve the network capacity in multihop networks, ranging from approaches that improve the efficiency of existing protocols, to approaches that use additional resources. In this dissertation, we propose to use additional frequency spectrum, as well as improve the efficiency of using existing frequency spectrum, for improving network capacity.

Widely used wireless technologies, such as IEEE 802.11, provision for multiple frequency-separated channels in the available frequency spectrum. Commercially available wireless network interfaces can typically operate over only one channel at a time. Due to cost and complexity constraints, the total number of interfaces at each node is expected to be fewer than the total number of channels available in the network. Under this scenario with fewer interfaces per node than channels, several challenges have to be addressed before all the channels can be utilized.

In this dissertation, we have established the asymptotic capacity of multichannel wireless networks with varying number of channels and interfaces. Capacity analysis has shown that it is feasible to effectively use a large number of channels even with only a few interfaces per node. Based on insights from the capacity analysis, we have developed a new interface management protocol and a new routing protocol to exploit multiple channels. Detailed simulation-based evaluations of the protocols have shown that multiple channels can be used to significantly enhance network capacity, even if only a few interfaces per node are available. We have implemented the protocols in a testbed to demonstrate the feasibility of using the protocols in practice. As part of the implementation, we have developed generic architectural support for using multiple channels, which may be of use in implementing other multichannel protocols as well.

To father and mother

ACKNOWLEDGMENTS

I would like to thank my adviser, Prof. Nitin Vaidya, for all his support and guidance during the course of my doctoral research. I thank Professors Jennifer Hou, Robin Kravets, and P. R. Kumar for serving on my dissertation committee. I would like to acknowledge National Science Foundation and Vodafone-US Foundation for financially supporting my research.

I would like to acknowledge the assistance received over the years from the members of my research group. I thank Lila Rhoades, and other staff at the Coordinated Science Laboratory, for providing administrative assistance.

The support of several friends has made graduate school a memorable experience. I would like to specially thank Shravan Gaonkar, Romit Roy Choudhury, Balaji Krishnan, Chandrakanth Chereddi, Arshad Ahmed, and Girish Baliga for all the great times we have spent together.

I am eternally grateful for the loving support of my parents and my brother. During my years in graduate school, the weekly conversations with them over the phone has always served to uplift my spirits.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Wireless networks have gained immense popularity over the last few years. The rapid growth may be attributed in part to popular wireless standards, such as IEEE 802.11 [1]. The predominant use of these wireless technologies has been in single hop local area networks that operate with infrastructure support. More recently, there has been significant research activity in building multihop wireless networks. The multihop paradigm can be used to build scalable networks at a lower cost than the single hop paradigm, and consequently, multihop communication has been employed in mesh networks, ad hoc networks, and sensor networks.

A key challenge in multihop wireless networks is to provision for sufficient network capacity to meet user requirements. Since wireless medium is a shared resource, the throughput obtained by a user goes down when the network density increases. Furthermore, an increasing number of popular applications require large bandwidth, for example, to support real-time audio and video downloads. As a result, network designers have to constantly strive to increase the network capacity.

Several approaches have been proposed to improve the network capacity in multihop networks, ranging from approaches that improve the efficiency of existing protocols, to approaches that use additional resources. In this thesis, we propose to use additional frequency spectrum, as well as improve the efficiency of using existing frequency spectrum, for improving network capacity. Additional spectrum may become available with changes in regulatory policies. For example, Federal Communications Commission, which manages spectrum allocation in United States, is considering re-allocating significant amount of spectrum for unlicensed use [2]. Although this is an encouraging trend toward more spectrum becoming available, there are few solutions for using already available frequency spectrum in multihop networks. Consequently, there is a need for

practical solutions that fully utilize currently available spectrum resources, while being flexible enough to incorporate additional frequency resources as they become available.

Widely used wireless technologies, such as IEEE 802.11, provision for multiple frequency-separated channels in the available frequency spectrum. Therefore, fully utilizing the available frequency spectrum requires the use of all the frequency-separated channels provisioned for in the standards. Multiple channels have been used in single hop infrastructure-based networks by assigning different channels to adjacent access points, thereby minimizing interference among access points. However, typical multihop wireless network configurations have used a single channel to ensure that all nodes in the network are connected. Therefore, to utilize all of the available spectrum, we develop new protocols that specifically promote the use of multiple channels.

Wireless hosts have typically been equipped with one wireless interface. Wireless interfaces that are currently available can tune to one channel at a time. Multiple channels can be simultaneously used by using multiple interfaces, and with reducing hardware costs [3], it is feasible to equip nodes with multiple interfaces. Nevertheless, in the foreseeable future it will remain expensive to equip every node with one dedicated interface per channel, because the number of available channels is already large, and the number of channels may further increase as and when additional frequency spectrum becomes available. Therefore, it is of practical interest to investigate the scenario wherein the *number of interfaces per node is smaller than the number of channels.*

Some solutions proposed in the past [4] for multichannel networks utilize only as many channels as interfaces per node. When the number of available channels is significantly larger than the number of available interfaces, this approach leads to inefficient utilization of spectrum. Commonly available commodity wireless interfaces can be switched from one channel to another, although switching incurs a non-negligible (but not too large) delay. This interface switching capability allows a node to dynamically switch and communicate on different channels over time. Therefore, even if a node does not have as many interfaces as channels, still a node could over time exploit all the available channels. Furthermore, a set of nodes in a neighborhood may collectively have enough interfaces to fully saturate all the channels. We propose to utilize all the available channels by using interface switching capability, in conjunction with the ability to intelligently distribute

interfaces in a neighborhood across available channels.

The focus of this thesis is to investigate the use of multiple channels to improve the network capacity in multihop wireless networks. Our emphasis is on the scenario wherein the number of interfaces per node is smaller than the number of available channels. The key contributions of the thesis are:

- Establishing the asymptotic capacity of multichannel wireless networks with varying number of channels and interfaces. The analysis showed that it is feasible to use a large number of channels with only a few interfaces per node. Insights from the analysis were used in designing practical solutions for using multiple channels.

- Design of protocols for multichannel wireless networks. We focused primarily on designing protocols at the link and routing layer under the assumption that multiple interfaces are available at each node. The protocols were shown to achieve significant performance improvements.

- Implementation of the proposed protocols in a realistic testbed. During the implementation process, we developed generic architectural support for using multiple channels, which may be of use in other multichannel implementations.

The rest of the thesis is organized as follows. In Chapter 2 we present the channel and interface model used in the thesis. In Chapter 3, we characterize the asymptotic network capacity of multichannel wireless networks with varying number of channels and interfaces. In Chapter 4, we specify protocol design requirements, identify the shortcomings of related work in meeting the requirements, and present our proposed protocol architecture. In Chapter 5, we present a link-layer interface management protocol for utilizing multiple channels. In Chapter 6, the interface management protocol is complemented with a new routing metric for effective utilization of multiple channels in multihop networks. In Chapter 7, we describe the implementation of the proposed protocols in a Linux-based testbed, and conclude in Chapter 8.

CHAPTER 2

CHANNEL AND INTERFACE MODEL

In this chapter, we first present the channel and interface model used in the paper. We then discuss some issues with the channels and interfaces that are available in practice.

2.1 Model

We assume that there are c channels available in the network. We use the term "channel" to refer to a part of the frequency spectrum with some specified bandwidth. The channels are assumed to be orthogonal, i.e., simultaneous transmissions are always possible on all orthogonal channels without interfering with each other. Under frequency division multiplexing, this assumption requires the frequency spectrum used by different channels to be nonoverlapping. Recent work [5] has noted the possibility of communicating over multiple channels without switching interfaces, provided the channels partially overlap. The protocols presented in this thesis can be extended to operate with partially overlapping channels as well.

Another assumption made in the thesis is that the c channels are homogeneous, i.e., all channels have the same transmission range and support the same set of data rates. The capacity analysis presented in the thesis continues to be valid even if channels are not homogeneous, provided the maximum difference in the ranges and maximum difference in rates over all the channels can be bounded by a constant. In addition, the multichannel protocols presented in this thesis work correctly even if the channels are not homogeneous. However, the protocols do not explicitly exploit heterogeneous channels to maximize performance. In Chapter 5, we discuss further the impact of channel heterogeneity on protocols.

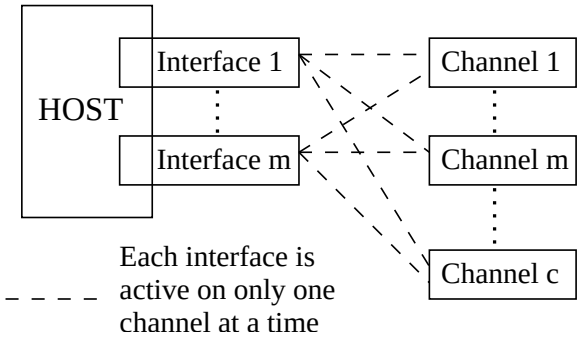

Figure 2.1: Channel and interface model.

We assume that every host in the network has m radio interfaces. However, the protocols in this thesis have been designed under a more general model, where each node i has m_i interfaces, and different nodes can potentially be equipped with a different number of interfaces. Each interface is assumed to be capable of either sending data or receiving data over only one channel at a time (half-duplex operation). Over time, interfaces are assumed to be capable of switching among any of the available c channels. However, switching an interface from one channel to another incurs a switching delay S, which may be non-negligible in practice. The protocols in this thesis have been designed to be operate effectively over a range of switching delays, thereby ensuring that the protocols are useful even if interface switching delays are different with new technologies and future hardware.

Figure 2.1 presents our channel and interface model. We use the notation (m, c)-network to refer to a network with m interfaces per node, and c channels. In this thesis, we study the capacity of a wireless network for different values of m and c. We then design protocols for the common scenario where there are fewer interfaces per host than channels available in the network ($m < c$).

2.2 Practical concerns

In this section, we first discuss the number of channels that are already available, as well as the number of channels that may become available in the near future. We then identify the number of orthogonal channels available for use with existing IEEE 802.11 hardware. We conclude with a study of the switching delay of currently available IEEE 802.11 hardware.

2.2.1 Spectrum to support large number of channels

In this thesis, we assume that channels are created by partitioning the available frequency spectrum (we justify this assumption later). Therefore, the number of available channels depends on the total spectrum that is available, as well as the bandwidth allocated to individual channels. Spectrum allocation is regulated by the federal communications commission (FCC) in the US, and traditionally, most of the spectrum in the lower frequency bands has been reserved for licensed users. In this thesis, we consider multihop networks that could be built with commercially available wireless hardware. Typically, commercially available wireless LAN hardware operate on license-exempt bands in the lower part of the frequency spectrum (e.g., 2.4 GHz, and 5 GHz spectrum). Therefore, the number of channels available depends on the amount of unlicensed spectrum that is available.

Recently, FCC is considering the possibility of allowing secondary users with cognitive radio capabilities to access licensed spectrum, provided such use does not interfere with the primary (licensed) user of spectrum. This portends the availability of large amount of spectrum in the lower frequency bands for unlicensed use in the near future. In addition, there has always been large amounts of unlicensed spectrum available in the higher frequency bands, but this spectrum was considered unsuitable for wireless LAN technologies. However, significant innovations in radio hardware technologies is opening the possibility of using spectrum in higher frequency bands. Figure 2.2 shows some of the spectrum that may potentially be used for unlicensed operation. The figure primarily shows unlicensed spectrum in the industrial, scientific and medical (ISM) bands, and this spectrum is available worldwide. In addition to spectrum in ISM bands, there is other spectrum for unlicensed use in the higher frequency bands (for example, around 7 GHz of spectrum is available in the 60 GHz band [6]). FCC is also considering opening up some spectrum in the 700 MHz band and the 3 GHz band for unlicensed use in the near future. To summarize, large amount of spectrum is available for unlicensed use, and this implies that a large number of channels will be available for use in the near future.

Even today, the number of channels available with existing wireless technologies is significantly large. For example, IEEE 802.11 provides for (in US) 12 nonoverlapping channels in the 5 GHz band, and 3 nonoverlapping channels in the 2.4 GHz band. The 12 channels in the 5 GHz band

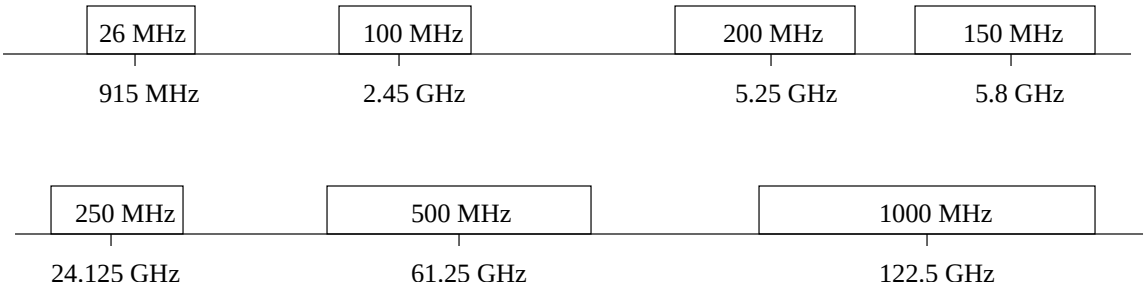

Figure 2.2: Spectrum available for unlicensed use (figure not to scale).

is already large compared to the number of interfaces that may be available at hosts. Therefore, there is a strong basis to focus on the scenario with fewer interfaces per host than channels.

2.2.2 Partitioning available spectrum into multiple channels

We assume that the available spectrum is divided into multiple non-overlapping channels. This assumption is true with existing wireless technologies, such as IEEE 802.11. Dividing the available spectrum into multiple channels, especially when large amount of spectrum is available, simplifies the design of interface hardware. Note that interfaces must be capable of operating over all the spectrum available in a channel. This implies that if all the available spectrum is used for a single channel, then interfaces have to be capable of sending/receiving data simultaneously over this entire spectrum. It is theoretically possible to design interfaces that can operate with large bandwidth channels, but it is difficult in practice to build such interfaces economically. The main reasons for the high cost of such interfaces are:

1. The available spectrum may not be contiguous. Therefore, interface hardware will have to be designed such that it is capable of transmission on several distinct contiguous frequency bands, while suppressing transmission on frequencies between any two bands. This may require multiple sets of filters, oscillators, etc., to handle each contiguous frequency band. In essence, the interface will have to be built as a combination of multiple simpler interfaces that are each capable of transmitting on one contiguous frequency band.

2. The maximum data rate achievable on a channel is dependent on the channel bandwidth and the signal-to-interference-noise (SINR). In an ideal setting, for a fixed amount of spectrum,

the aggregate data rate over all the channels is a constant, independent of the number of channels that are created out of the spectrum, provided the same SINR is maintained on each channel. However, the total noise power is directly proportional to the amount of spectrum that is available. Therefore, to achieve some fixed SINR, transmitted signal power has to be scaled linearly with the channel bandwidth. It is difficult to build interfaces, which require high-power transmitters, at a low cost (and with a small form-factor). Instead, if channel bandwidth is small, individual interfaces need to only support a low transmission power, while the total power output by all the interfaces over all channels in the network could still be high. In addition, the amount of path loss depends on the frequency of operation, with larger path losses at higher frequencies. Therefore, for a large bandwidth channel, the transmitted signal power may have to be varied across different parts of the spectrum used by the channel [7], to compensate for path loss variations, increasing hardware complexity.

3. Interfaces typically include digital signal processing hardware to encode the transmitted packets and decode the received packets. When the channel bandwidth is large (leading to high data rates), the signal processing requirements are proportionately larger [8]. This necessitates fast signal processing hardware (or multiple processing units), which could further increase hardware costs.

In addition to the complexities of building interfaces capable of handling large bandwidth channels, MAC protocol overheads are also higher with larger bandwidth channels [9]. Some of the MAC protocol overheads consume a constant amount of time per packet on the channel, independent of the channel bandwidth (e.g., contention slot length is dependent on the wireless propagation delay, and is independent of the channel bandwidth). Such overheads consume more resources per packet with larger channel bandwidth (as more data could have been sent in a given amount of overhead time over a wider channel than over a narrower channel). Therefore, this also motivates partitioning available spectrum into multiple channels.

While the above arguments motivate partitioning the available spectrum into multiple channels, having too many channels is also inefficient. When spectrum is divided into multiple channels, "guard" bands must be created between channels to prevent cross-channel interference. Typically,

the amount of guard spectrum required per channel is independent of the channel bandwidth (it depends on the quality of the frequency filters), which implies that the total guard spectrum increases linearly with number of channels. This suggests that having too many channels is also inefficient, because the guard spectrum overheads could become significant. In practice, there is some optimal number of channels that maximizes channel utilization, and this optimal number depends in part on the capabilities of the available interfaces (which in turn depends on cost tradeoffs). With current interface hardware capabilities, it seems likely that several channels will be available for use.

2.2.3 Channel orthogonality

We evaluate the protocols proposed in this thesis on a testbed that uses IEEE 802.11 interfaces. As we discussed in the previous section, IEEE 802.11 offers 12 nonoverlapping channels in the 5 GHz band[1]. With commercially available IEEE 802.11 interfaces, when a signal is transmitted on a channel, some of the transmitted signal leaks on to adjacent channels [4]. Less power is leaked on to channels that are farther away in the spectrum. Therefore, the leaked signal power is only strong enough to interfere with transmissions on channels that are close (in frequency) to the transmitting channel. The power of the leaked signal also attenuates with distance. Therefore, the leaked signal will not interfere at a receiver that is sufficiently separated in distance from the transmitter. Consequently, even if some signal power leaks on to adjacent channels, it may not interfere with transmissions on adjacent channels, if interfaces are sufficiently separated from each other.

When a single host is equipped with multiple interfaces, one interface might be transmitting on a channel, while data is being received on another interface on an adjacent channel. Since it is likely that multiple interfaces at the same host are close to each other (in our testbed, interfaces are separated by a distance of less than six inches), the leaked transmitted signal could interfere with parallel receptions at that host. One way of overcoming this is to develop protocols that ensure that at any time all the interfaces at a node are all either receiving data or idle, or all interfaces are either sending data or idle. Such a separation will ensure that one interface at a

[1]These channels are each 20 MHz wide, and their center frequencies are separated by 20 MHz, so they do not overlap in frequency.

node will not be sending data while another interface is receiving data. The key challenge with this approach is the need for synchronization among different interfaces. One of our goals in this thesis was to develop protocols that work with off-the-shelf hardware, which do not support the desired synchronization. Therefore, we used an alternate approach that does not preclude simultaneous transmissions and receptions. Instead, to ensure that simultaneous transmissions and receptions at a host do not interfere with each other, we use only a subset of channels that are sufficiently separated in frequency from each other. This ensures that the channels are truly orthogonal, i.e., any transmissions on different channels do not interfere with each other.

The number of orthogonal channels available in the network depends on several factors: the specific interface cards used (which affects the signal leakage), the minimum distance between a receiving interface and its nearest interfering transmitter (which affects the interference signal strength), and the maximum distance between any receiving interface and its corresponding transmitting interface (which affects the received signal strength). Future hardware may employ better channel filters to reduce adjacent channel interference. In addition, it may be possible to place multiple interfaces on a host with more separation than what is currently possible, and add shielding material between interfaces, to reduce adjacent channel interference. Using a combination of such approaches may allow many more channels to be simultaneously used.

In simulation studies, we account for the possibility that interference among adjacent nonoverlapping channels may reduce the number of channels that are orthogonal, by evaluating the protocols for different number of channels. For example, our experiments (discussed below) show that 5 out of 12 channels can be used orthogonally in the 5 GHz band (IEEE 802.11a) with our testbed hardware. Therefore, simulation results study the performance of protocols with 5 channels. In addition, improved hardware in the future may allow all the 12 nonoverlapping channels provisioned for in IEEE 802.11 to be used orthogonally. Therefore, our simulations also study the performance of protocols with 12 channels.

Orthogonal channel measurements

We studied the number of orthogonal channels available for use with the testbed hardware available in our lab. The hosts in our lab are simple computers from Soekris [10], equipped with two radio interfaces based on Atheros 5212 chipset [11]. Here we present one set of measurements that show the availability of five orthogonal channels in the IEEE 802.11a band. More detailed results are available in [12].

The 802.11a specification provides for 12 nonoverlapping channels, 4 each in the U-NII lower band (5.15 to 5.25 GHz), U-NII middle band (5.25 to 5.35 GHz), and U-NII upper band (5.725 to 5.825 GHz). Each of these channels is 20 MHz wide. The channels in the lower band are numbered 36, 40, 44, and 48; the channels in the middle band are numbered 52, 56, 60, and 64; and the channels in upper band are numbered 149, 153, 157, and 161. Observe that there is a significant separation between the middle band and the upper band. As a result, transmissions on channels in the lower and middle band do not interfere with transmissions on channels in the upper band.

For the experimental evaluation, two nodes, named A and B, were successively placed 20 ft, 40 ft, 60 ft, and 80 ft apart. One constant bit rate (CBR) flow was set up from the first interface of node A to the first interface of node B, with both interfaces tuned to a common channel. Simultaneously, node B initiated a ping flood to a broadcast address[2] (at a rate large enough to saturate the channel), using its second interface, to create adjacent channel interference. For lower and middle band measurements, the first interface of node B was placed on channel 52, and the second interface of node B was tuned to different channels ranging from 36 to 64. For the upper band measurements, the first interface of node B was placed on channel 149, and the second interface was tuned to channels ranging from 149 to 161. For comparison, one set of experiments were conducted when the broadcast ping traffic was disabled, thereby avoiding adjacent channel interference.

Figure 2.3 plots the throughput of the CBR flow when the channel of the interfering ping traffic is varied. When the broadcast ping was disabled, the CBR flow throughput was 5.3 Mbps. When the broadcast ping traffic is enabled, adjacent channel interference may reduce the CBR

[2]Packets were sent out on a broadcast address to allow ping to continue even if no response was received.

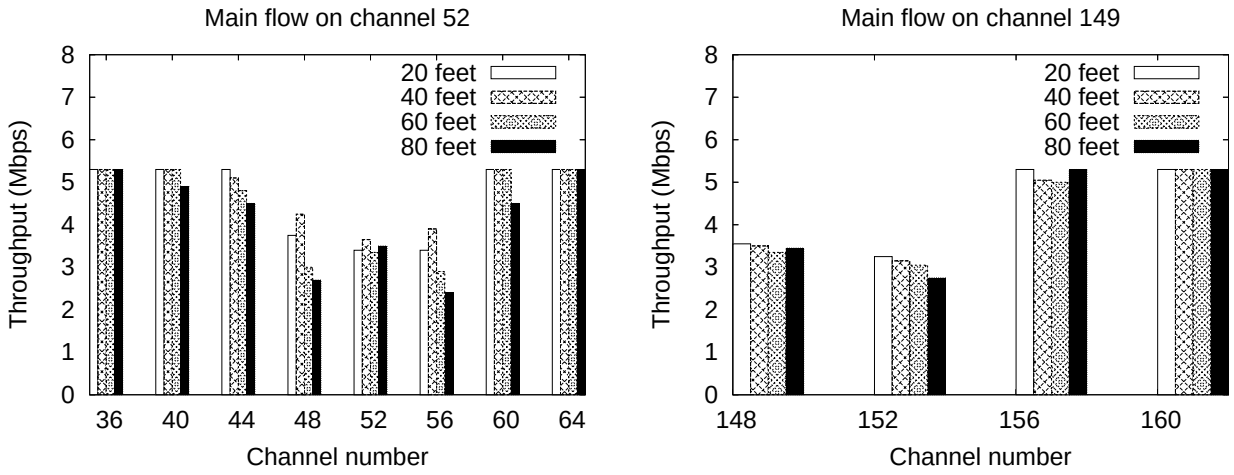

Figure 2.3: CBR throughput versus channel used by interfering ping traffic. Throughput with no interference is 5.3 Mbps.

throughput. We can see from the figure that in the lower and middle bands, channels 36 and 64 do not interfere with channel 52, since CBR throughput is not affected by broadcast ping traffic. Therefore, channels 36, 52, and 64 are orthogonal. In the upper band, we observe that channel 161 does not interfere with channel 149. Therefore, channels 149 and 161 are orthogonal. Also, we have verified that channels in the lower and middle frequency bands do not interfere with the channels in the upper frequency band. Hence, five orthogonal channels are available with the hardware used in our testbed.

2.2.4 Interface switching delay

The ability to switch an interface from one channel to another is a key property we exploit to utilize all the available channels when the number of interfaces available is significantly smaller than the number of available channels. Switching an interface from one channel to another requires a change in the frequency of operation in the hardware, and therefore incurs a delay. In addition, there is a delay incurred before the switch request by the device driver is acted upon by the firmware. The delay for switching an interface also depends on where the switch request is invoked from; in general, requests from userspace incur larger delay than requests from the kernel. In this thesis, we develop a channel abstraction module in the Linux kernel to support multichannel operation, and enable fast interface switching. Our notion of switching delay is the time incurred from when

a channel switch request is issued to the driver to when the driver is ready to send data over the new channel (after completing the switch).

The time required to switch channels for some of the commercially available hardware is of the order of a few hundred microseconds. However, the time required for the driver to complete a channel switch is significantly larger, and can be as large as tens of milliseconds with certain hardware (and their associated drivers). In our testbed, we use wireless interfaces based on the Atheros chipset [11] along with the madwifi driver [13]. The IEEE 802.11 standard was not designed for multichannel operation, and incurs high protocol overheads after a channel switch. We have disabled some of these overheads in the madwifi driver (details are in [12]), and this reduces the interface switching delay.

Our experiments with the modified madwifi driver [12] show that the switching delay is 5 ms for the interfaces used in our testbed. We have also identified certain other optimizations that may be possible with madwifi driver to reduce the switching delay to 2 ms. The benefits of utilizing multiple channels by switching interfaces has become evident only recently, and wireless interface manufacturers do not seem to have yet made explicit efforts to reduce the interface switching delay. We believe that future hardware designed to support interface switching will have significantly lower switching delay.

As discussed earlier, the protocols in this thesis have been designed to operate effectively over a range of switching delays. Most of our protocol simulations set the switching delay to 1 ms, though we have some results which study the performance with a switching delay of 5 ms (delay measured on our testbed), and with a switching delay of 100 μs (delay that may be achievable in the future).

CHAPTER 3

CAPACITY OF MULTICHANNEL NETWORKS

In this chapter, we study the asymptotic capacity of multichannel wireless networks with varying number of interfaces. Past research on wireless network capacity [14, 15] has typically considered wireless networks with a single channel, although the results are applicable to a wireless network with multiple channels as well, provided that at each node there is a dedicated interface per channel. With a dedicated interface per channel, a node can use all the available channels simultaneously. However, as we discussed earlier, the number of available channels in a wireless network can be fairly large, and it may not be feasible to have a dedicated interface per channel at each node. When nodes are not equipped with a dedicated interface per channel, then *capacity degradation* may occur, compared to using a dedicated interface per channel.

In this chapter, we characterize the impact of number of channels and interfaces per node on the network capacity [16], and show that in certain scenarios, even with only a single interface per node, there is no capacity degradation. This implies that it may be possible to build capacity-optimal multichannel networks with as few as one interface per node. Our initial analysis assumes that the interface switching delay is zero, which may not be valid in practice. Nevertheless, even when interface switching delay is accounted for, capacity-optimal performance can be achieved by using only a few interfaces per node. In addition, if each node has a single interface that is never switched, then there is a degradation in the network capacity. However, with only two interfaces per node, there is no capacity degradation even if the interfaces are not switched.

The rest of this chapter is organized as follows. We present the channel and network model, as well as an overview of the main results in Section 3.1. We present related work in Section 3.2. In Section 3.3, we establish the capacity of multichannel networks under arbitrary network setting.

Section 3.4 establishes the capacity of multichannel networks under random network setting. Section 3.5 characterizes the impact of interface switching delay, and Section 3.6 studies the capacity when interfaces do not switch at all. We conclude this chapter in Section 3.7.

3.1 Introduction

In this section, we first define the channel and network model, and then provide an overview of results.

3.1.1 Channel and interface model

We consider a static wireless network containing n nodes. We restate here the channel and interface model from Chapter 2 for completeness. In our model there are c channels, and we assume that every node is equipped with m interfaces, $1 \leq m \leq c$. We assume that an interface is capable of transmitting or receiving data on any one channel at a given time. We use the notation (m, c)-network to refer to a network with m interfaces per node, and c channels.

We define two channel models to represent the data rate supported by each channel:

Channel Model 1: In model 1, we assume that the total data rate possible by using all channels is W. The total data rate is divided equally among the channels, and therefore the data rate supported by any one of the c channels is W/c. This was the channel model used by Gupta and Kumar [14], and we primarily use this model in our analysis. In this model, as the number of channels increases, each channel supports a smaller data rate. This model is applicable to the scenario where the total available bandwidth is fixed, and new channels are created by partitioning existing channels.

Channel Model 2: In model 2, we assume that each channel can support a fixed data rate of W, independent of the number of channels. Therefore, the aggregate data rate possible by using all c channels is Wc. This model is applicable to the scenario where new channels are created by utilizing additional frequency spectrum.

The capacity results are derived using channel model 1. However, all the derivations are applicable for channel model 2 as well, and the results for model 2 are obtained by replacing W in the results of model 1 by Wc.

3.1.2 Network and traffic model

We study the capacity of static multichannel wireless networks under the two settings introduced by Gupta and Kumar [14].

Arbitrary Networks: In the arbitrary network setting, the location of nodes, and traffic patterns can be controlled. Since any suitable traffic pattern and node placement can be used, the bounds for this scenario are applicable to any network. Therefore, the arbitrary network bounds may be viewed as the *best case* bounds on network capacity, as the bounds are applicable to all networks. The network capacity is measured in terms of "bit-meters/sec" (originally introduced by Gupta and Kumar [14]). The network is said to transport one "bit-meter/sec" when one bit has been transported across a distance of one meter in one second.

Random Networks: In the random network setting, node locations are randomly chosen, i.e. independently and uniformly chosen, on the surface of an unit torus. Each node sets up one flow to a randomly chosen destination. The network capacity is defined to be the aggregate throughput over all the flows in the network, and is measured in terms of bits/sec.

3.1.3 Definitions

We use the following notation [17] to represent asymptotic bounds:

1. $f(n) = O(g(n))$ implies that there exists some constant d and integer N such that $f(n) \leq dg(n)$ for $n > N$.
2. $f(n) = o(g(n))$ implies that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
3. $f(n) = \Omega(g(n))$ implies $g(n) = O(f(n))$.
4. $f(n) = \omega(g(n))$ implies $g(n) = o(f(n))$.

5. $f(n) = \Theta(g(n))$ implies $f(n) = O(g(n))$ and $g(n) = O(f(n))$.

6. $MIN_O\left(f(n), g(n)\right)$ is equal to $f(n)$, if $f(n) = O(g(n))$, else, is equal to $g(n)$.

The bounds for random networks hold *with high probability (whp)*. In this chapter, *whp* implies "with probability 1 when $n \to \infty$."

3.1.4 Main Results

Gupta and Kumar [14] have shown that in an arbitrary network, network capacity scales as $\Theta\left(W\sqrt{n}\right)$ bit-meters/sec, and in a random network, the network capacity scales as $\Theta\left(W\sqrt{\frac{n}{\log n}}\right)$ bits/sec. Under the channel model 1, which was the model used by Gupta and Kumar [14], the capacity of a network with a single channel and one interface per node (that is, a $(1,1)$-network in our notation) is equal to the capacity of a network with c channels and $m = c$ interfaces per node (that is, a (c,c)-network). This equivalence arises because the c interfaces can operate in parallel over channels of data rate $\frac{W}{c}$ to mimic the operation of one interface operating over a channel of data rate W (this is formally proved in Lemma 1). Furthermore, under both channel models, the capacity of a (c,c)-network is at least as large as the capacity of a (m,c)-network, when $m \leq c$ (this is trivially true, by not using $c - m$ interfaces in the (c,c)-network).

In the results presented in this chapter, we capture the impact of using fewer than c interfaces per node by establishing the *loss in capacity*, if any, of a (m,c)-network in comparison to a (c,c)-network. We show that there are distinct capacity regions, the boundaries of which depend on the ratio $\frac{c}{m}$, and not on the exact values of either c or m. Here we present an overview of the main results, under channel model 1.

1. Results for arbitrary network: The network capacity of a (m,c)-network has two regions (see Figure 3.1) as follows (from Theorem 2 and Theorem 4):

1. When $\frac{c}{m}$ is $O(n)$, the network capacity is $\Theta\left(W\sqrt{\frac{nm}{c}}\right)$ bit-meters/sec (segment A-B in Figure 3.1). Compared to a (c,c)-network, there is a capacity loss by a factor of $1 - \sqrt{\frac{m}{c}}$.

2. When $\frac{c}{m}$ is $\Omega(n)$, the network capacity is $\Theta\left(W\frac{nm}{c}\right)$ bit-meters/sec (line B-C in Figure 3.1). In this case, there is a larger capacity degradation than case 1, as $\frac{nm}{c} \leq \sqrt{\frac{nm}{c}}$ when $\frac{c}{m} \geq n$.

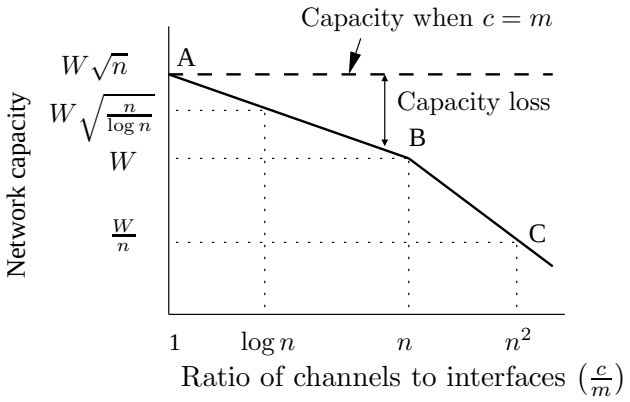

Figure 3.1: Impact of number of channels on capacity scaling in arbitrary networks (figure is not to scale).

Therefore, there is always a capacity loss in arbitrary networks whenever the number of interfaces per node is fewer than the number of channels.

2. Results for random network: The network capacity of a (m, c)-network has three regions (see Figure 3.2) as follows (from Theorem 6 and Theorem 9):

1. When $\frac{c}{m}$ is $O(\log n)$, network capacity is $\Theta\left(W\sqrt{\frac{n}{\log n}}\right)$ bits/sec (segment D-E in Figure 3.2). In this case, *there is no loss* compared to a (c, c)-network. Hence, in many practical scenarios where c may be constant or small, *a single interface per node suffices.*

2. When $\frac{c}{m}$ is $\Omega(\log n)$ and also $O\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, the network capacity is $\Theta\left(W\sqrt{\frac{nm}{c}}\right)$ bits/sec (segment E-F in Figure 3.2). In this case, there is some capacity loss. Furthermore, in this region, the capacity of a (m, c)-random network *is the same* as that of a (m, c)-arbitrary network (segment E-F in Figure 3.2 overlaps part of segment A-B in Figure 3.1), implying that "randomness" does not incur a capacity penalty.

3. When $\frac{c}{m}$ is $\Omega\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, the network capacity is $\Theta\left(\frac{Wnm\log\log n}{c\log n}\right)$ bits/sec (line F-G in Figure 3.2). In this case, there is a larger capacity degradation than case 2. Furthermore, in this region, the capacity of a (m, c)-random network *is smaller than* that of a (m, c)-arbitrary network, in contrast to case 2.

3. Impact of switching delay: The results presented above are derived under the assumption that there is no delay in switching an interface from one channel to another. However, we show that in

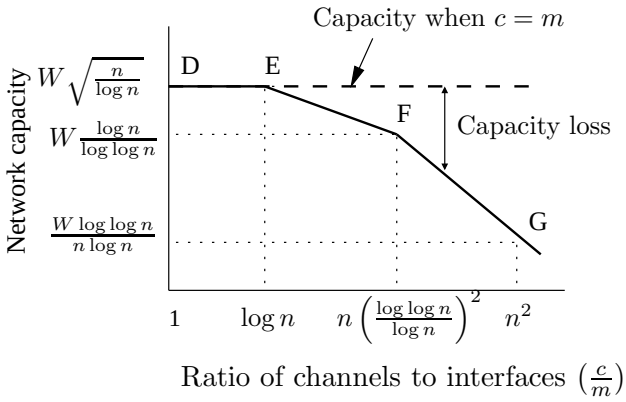

Figure 3.2: Impact of number of channels on capacity scaling in random networks (figure is not to scale).

a random network with up to $O(\log n)$ channels, even if interface switching delay is considered, the network capacity is not reduced, provided a few additional interfaces are provisioned for at each node. This implies that it may be possible to hide the interface switching delay by using extra interfaces in conjunction with carefully designed routing and transmission scheduling protocols.

4. Impact of keeping interfaces fixed: In practice, protocol implementation can be simplified if interfaces are fixed to channels. We show that if every node has a single interface, and the interface is never switched (after initially assigning the interface to some channel), then there is a loss in the network capacity. This loss in capacity can be avoided, by having only two interfaces per node, even if the interfaces do not switch.

3.2 Related work

In their seminal work, Gupta and Kumar [14] established the capacity of ad hoc wireless networks. The results are applicable to single channel wireless networks, or multichannel wireless networks where every node has a dedicated interface per channel. We extend the results of Gupta and Kumar to those multichannel wireless networks where nodes may not have a dedicated interface per channel, and we also consider the impact of interface switching delay on network capacity.

Grossglauser and Tse [15] showed that mobility can improve network capacity, though at the cost of increased end-to-end delay. Subsequently, other research [18, 19] has analyzed the trade-

off between delay and capacity in mobile networks. Gamal et al. [17] characterize the optimal throughput-delay trade-off for both static and mobile networks. In this thesis, we adapt some of the proof techniques presented by Gamal et al. [17] to the multichannel capacity problem. Lin et al. [20, 21] also study the throughput-delay trade-off in wireless networks.

Recent results have shown that the capacity of wireless networks can be enhanced by introducing infrastructure support [22–24]. Other approaches for improving network capacity include the use of directional antennas [25], and the use of unlimited bandwidth resources (UWB), albeit with power constraints [26, 27]. Li et al. [28] have used simulations to evaluate the capacity of multichannel networks based on IEEE 802.11. Other research on capacity is based on considerations of alternate communication models [29–31], but do not consider the multichannel scaling problem.

Kodialam et al. [32] have studied the throughput achievable in a multichannel multi-interface network by using constrained optimization techniques. Their work is applicable to scenarios where the network topology and traffic patterns are known a priori. Alicherry et al. [33] have considered a similar multichannel multi-interface problem, but for a restricted class of mesh networks (where all traffic is directed toward gateway nodes). Zhang et al. [34] have studied the benefits of jointly optimizing both routing and scheduling in multichannel multi-interface networks. All these works are well suited for network planning, but are less useful in understanding scaling properties of the network.

3.3 Capacity results for arbitrary networks

We assume that all nodes transmit at the same data rate, and use the same transmission power. We model the impact of interference by using the protocol model proposed by Gupta and Kumar [14]. The transmission from a node i to a node j on some channel x is successful, if for every other node k simultaneously transmitting on channel x, the following condition holds:

$$d(k,j) \geq (1+\Delta)d(i,j),\ \Delta > 0$$

where $d(i,j)$ is the distance between nodes i and j, and Δ is a "guard" parameter to ensure that any other concurrently transmitting nodes are sufficiently farther away from the receiver to prevent excessive interference.

It is shown in [14] that the protocol model is equivalent to an alternate physical model that is based on received Signal-to-Interference-Noise Ratio (SINR) (when path loss exponent is greater than 2). Therefore, the results in this thesis are applicable under the physical model as well. We do not consider other physical layer characteristics such as channel fading in our analysis. We derive the capacity results for arbitrary and random networks under the assumption that there is no switching delay. We extend our model to consider the impact of switching delay in Section 3.5.

In an arbitrary network, the location of nodes, and traffic patterns can be controlled. Recall that the network is said to transport one "bit-meter/sec" when one bit has been transported across a distance of one meter in a second. The network capacity of an arbitrary network is measured in terms of bit-meters per second, instead of bits per second. The bit-meters/sec metric is a measure of the "work" that is done by the network in transporting bits. In the case of random networks, the average distance traveled by any bit is $\Theta(1)$, and therefore the "bit-meters/sec" and "bits/sec" capacity is of the same order.

We assume that n nodes can be located anywhere on the surface of a torus of unit area, as in [17]. The assumption of a torus enables us to avoid technicalities arising out of edge effects, but the results are applicable for nodes located on an unit square as well. We first establish an upper bound on the network capacity of arbitrary networks, and then construct a network to prove that the bound is tight.

3.3.1 Upper bound on capacity

The capacity of multichannel arbitrary networks is limited by two constraints (described below), and each of them is used to obtain a bound on the network capacity. The minimum of the two bounds (the bounds depend on ratio between the number of channels c and the number of interfaces m) is an upper bound on the network capacity. While there may be other constraints on capacity as well, the constraints we consider are sufficient to provide a tight bound. Later in this section, we

will present a lower bound that matches the upper bound established by the two constraints, which validates our claim that the constraints are tight. We derive the bounds under channel model 1, although the derivation can be applied to channel model 2 as well[1].

Constraint 1 – Interference constraint: The capacity of any wireless network is constrained by interference. Since the wireless channel is a shared medium, under the assumed protocol model of interference, two nodes simultaneously receiving a packet from two different transmitters must have a minimum separation between them, which depends on Δ. This implies that there is a bound on the maximum number of simultaneous transmissions in the network. Based on this observation, using the proof techniques presented in [14] with some modifications to account for multiple interfaces and channels, one bound on the network capacity is $O\left(W\sqrt{\frac{nm}{c}}\right)$ bit-meters/sec. The detailed derivation is below in Theorem 1.

Theorem 1. *An upper bound on the capacity of a (m, c)-network under the arbitrary network setting is $O\left(W\sqrt{\frac{nm}{c}}\right)$ bit-meters/sec under channel model* 1.

Proof. We prove the result under channel model 1. The proof is based on a proof in [14]. We assume that nodes are synchronized, and slotted transmissions of duration τ are used. We assume that each source node originates λ bits/sec. Let the average distance between source and destination pairs be $\bar{L}$. Therefore, the capacity of the network is $\lambda n\bar{L}$ bit-meters/sec.

We consider any time period of length one second. In this time interval, consider a bit b, $1 \leq b \leq \lambda n$. We assume that bit b traverses $h(b)$ hops on the path from its source to its destination, where the h-th hop traverses a distance of r_b^h. Since the distance traversed by a bit from its source to its destination is at least equal to the length of the line joining the source and the destination, by summing over all bits we obtain,

$$\sum_{b=1}^{\lambda n}\sum_{h=1}^{h(b)} r_b^h \geq \lambda n\bar{L} \tag{3.1}$$

Let us define H to be the total number of hops traversed by all transmitted bits in a second, i.e., $H = \sum_{b=1}^{\lambda n} h(b)$. Therefore, the number of bits transmitted by all nodes in a second (including bits relayed) is equal to H. Since each node has m interfaces, and each interface transmits over a

[1]Recall that the results under channel model 2 can be obtained by replacing W with Wc in the results derived under channel model 1.

channel with rate W/c (assuming channel model 1), the total bits that can be transmitted by all nodes over all interfaces is at most $\frac{Wmn}{2c}$ (Transporting a bit across one hop requires *two* interfaces, one each at the transmitting and the receiving nodes). Hence, we have,

$$H \leq \frac{Wmn}{2c} \tag{3.2}$$

Under the protocol model, a transmission over a hop of length r is successful only if there is no other node transmitting within a distance of $(1+\Delta)r$ of the receiver. Suppose node A is transmitting a bit to node B, while node C is simultaneously transmitting a bit to node D, and both the transmissions are over a common channel. Then, using the protocol interference model, both transmissions are successful only if

$$d(C,B) \geq (1+\Delta)d(A,B)$$

$$d(A,D) \geq (1+\Delta)d(C,D)$$

Adding the above two expressions together, and applying triangle inequality, we obtain,

$$d(B,D) \geq \frac{\Delta}{2}(d(A,B)+d(C,D))$$

This implies that the receivers of two simultaneous transmissions have to be separated by a distance proportional to the distance from their senders. This may be viewed as each hop consuming a disk of radius $\frac{\Delta}{2}$ times the length of the hop around each receiver. Since the area "consumed" on each channel is bounded above by the area of the domain (1 sq meter), summing over all channels (which can in total potentially transport W bits) we have the constraint,

$$\sum_{b=1}^{\lambda n}\sum_{h=1}^{h(b)} \frac{\pi\Delta^2}{4}(r_b^h)^2 \leq W \tag{3.3}$$

which can be rewritten as,

$$\sum_{b=1}^{\lambda n}\sum_{h=1}^{h(b)} \frac{1}{H}(r_b^h)^2 \leq \frac{4W}{\pi\Delta^2 H} \tag{3.4}$$

Since the expression on the left hand side is convex, we have,

$$(\sum_{b=1}^{\lambda n}\sum_{h=1}^{h(b)}\frac{1}{H}r_b^h)^2 \leq \sum_{b=1}^{\lambda n}\sum_{h=1}^{h(b)}\frac{1}{H}(r_b^h)^2 \tag{3.5}$$

Therefore, from (3.4) and (3.5),

$$\sum_{b=1}^{\lambda n}\sum_{h=1}^{h(b)} r_b^h \leq \sqrt{\frac{4WH}{\pi\Delta^2}} \tag{3.6}$$

Substituting for H from (3.2), and using (3.1) we have,

$$\lambda n\bar{L} \leq W\sqrt{\frac{2mn}{\pi\Delta^2 c}} \tag{3.7}$$

This proves that the network capacity of an arbitrary network is $O\left(W\sqrt{\frac{nm}{c}}\right)$ bit-meters/sec under channel model 1. □

Constraint 2 – Interface bottleneck constraint: The capacity of a wireless network is also constrained by the maximum number of bits that can be transmitted simultaneously over all interfaces in the network. Since each node has m interfaces, there are a total of mn interfaces in the (m, c)-network. Each interface can transmit at a rate of $\frac{W}{c}$ bits/sec. Also, the maximum distance a bit can travel in the network is $O(1)$ meters. Hence, the total network capacity is at most $O\left(W\frac{nm}{c}\right)$ bit-meters/sec. This bound is tight when $\frac{c}{m}$ is $\Omega(n)$.

Combining the two constraints, the network capacity is $O\left(MIN_O\left(W\sqrt{\frac{nm}{c}}, W\frac{nm}{c}\right)\right)$ bit-meters/sec, under channel model 1. Therefore, we have the following theorem on the network capacity of arbitrary networks (Figure 3.1 has a pictorial representation).

Theorem 2. *The upper bound on the capacity of a (m, c)-arbitrary network under channel model 1 is as follows:*

1. *When $\frac{c}{m}$ is $O(n)$, network capacity is $O\left(W\sqrt{\frac{nm}{c}}\right)$ bit-meters/sec.*
2. *When $\frac{c}{m}$ is $\Omega(n)$, network capacity is $O\left(W\frac{nm}{c}\right)$ bit-meters/sec.*

The result for channel model 2 can be similarly derived, and is given by:

Theorem 3. *The upper bound on the capacity of a (m, c)-arbitrary network under channel model 2 is as follows:*

1. *When $\frac{c}{m}$ is $O(n)$, network capacity is $O\left(W\sqrt{nmc}\right)$ bit-meters/sec.*
2. *When $\frac{c}{m}$ is $\Omega(n)$, network capacity is $O\left(Wnm\right)$ bit-meters/sec.*

The network capacity of a (c, c)-network is $O\left(W\sqrt{n}\right)$ bit-meters/sec under channel model 1, which was the result obtained by Gupta and Kumar [14]. When fewer interfaces are available, there is a capacity degradation by at least a factor of $1-\sqrt{\frac{m}{c}}$. Intuitively, the capacity degradation arises because the total bits that can be simultaneously transmitted decreases.

3.3.2 Constructive lower bound

In this section, we construct a network to establish a lower bound on the network capacity. The lower bound matches the upper bound, implying that the bounds are tight. We first establish two results that we use in the rest of the chapter. The results are proved under channel model 1, but hold for channel model 2 as well.

Lemma 1. *Suppose m, c, $\tilde{c}$ are positive integers such that $\tilde{c} = \frac{c}{m}$. Then, a (m, c)-network can support at least the capacity supported by a $(1, \tilde{c})$-network, under channel model 1.*

Proof. Consider a (m, c)-network. We group the c channels into $\tilde{c}$ groups (numbered from 1 to $\tilde{c}$), with m channels per group as shown in Figure 3.3. Specifically, channel group i, $1 \leq i \leq \tilde{c}$, contains all channels j such that $(i-1)m + 1 \leq j \leq im$.

Assume that time on the channels is divided into slots of duration τ. Consider any slot s. Suppose a node X in the $(1, \tilde{c})$-network has its interface on some channel i, $1 \leq i \leq \tilde{c}$, in slot s. We simulate this behavior in the (m, c)-network by assigning the m interfaces of X in the slot s to the m channels in the channel group i. In this fashion, in any slot, the m interfaces of any node in the (m, c)-network are mapped to a channel group. The aggregate data rate of each channel group is $Wm/c = W/\tilde{c}$ (since $c = m\tilde{c}$). Therefore, a channel group in the (m, c)-network can support the same data rate as a channel in the $(1, \tilde{c})$-network. This mapping allows the (m, c)-network to mimic the behavior of $(1, \tilde{c})$-network; the $W\tau/\tilde{c}$ bits sent on some channel in any time slot s in

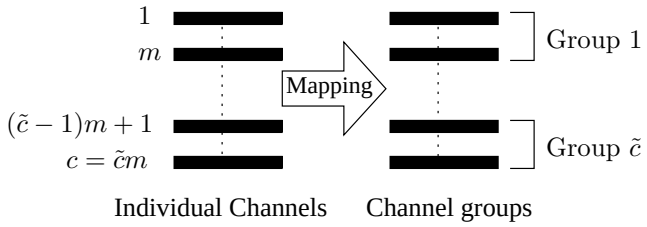

Figure 3.3: Lemma 1 construction: Forming $\tilde{c}$ channel groups, with m channels per group, in a (m, c)-network.

the $(1, \tilde{c})$-network can be simulated by sending $W\tau/c$ bits (in the same slot s) on each of the m channels in the corresponding channel group of the (m, c)-network. Hence, a (m, c)-network can support the capacity of a $(1, \tilde{c})$ network, when $c = m\tilde{c}$. □

Lemma 2. *Suppose m and c are positive integers. Then, a (m, c)-network can support at least $\frac{1}{2}$ the capacity supported by a $\left(1, \lfloor \frac{c}{m} \rfloor\right)$-network, under channel model* 1.

Proof. Suppose $\lfloor \frac{c}{m} \rfloor = \frac{c}{m}$. Then the result directly follows from the previous lemma. Otherwise, $m < c$, and we use $c' = m \lfloor \frac{c}{m} \rfloor$ of the channels in the (m, c)-network, and ignore the rest of the channels. This can be viewed as a (m, c')-network, with a total data rate of $W' = W \frac{m}{c} \lfloor \frac{c}{m} \rfloor$ (as each channel supports $\frac{W}{c}$ bits/sec). Using Lemma 1, a (m, c')-network with total data rate of W' can support at least the capacity of a $\left(1, \lfloor \frac{c}{m} \rfloor\right)$-network with total data rate of W'. However, when $W' < W$, the (m, c')-network with total data rate W' can achieve only a fraction $\frac{W'}{W}$ of the capacity of a $\left(1, \lfloor \frac{c}{m} \rfloor\right)$-network with total data rate W (instead of W'). Now,

$$\begin{aligned}
\frac{W'}{W} &= \frac{m}{c} \left\lfloor \frac{c}{m} \right\rfloor \\
&= \frac{\lfloor \frac{c}{m} \rfloor}{\frac{c}{m}} \\
&\geq \frac{\lfloor \frac{c}{m} \rfloor}{\lfloor \frac{c}{m} \rfloor + 1}, \text{ since } \frac{c}{m} \leq \left\lfloor \frac{c}{m} \right\rfloor + 1 \\
&\geq \frac{1}{2}, \text{ since } \left\lfloor \frac{c}{m} \right\rfloor \geq 1
\end{aligned}$$

Hence, a (m, c)-network can support at least $\frac{1}{2}$ the capacity supported by a $\left(1, \lfloor \frac{c}{m} \rfloor\right)$ network. Hence, asymptotically, a (m, c)-network has the same order of capacity as a $\left(1, \lfloor \frac{c}{m} \rfloor\right)$-network. □

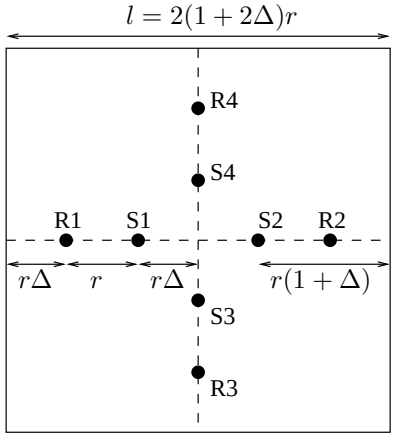

Figure 3.4: The placement of nodes within a cell. There are k nodes at each of the labeled positions.

We now provide the construction to establish that a capacity of $\Omega\left(MIN_O\left(W\sqrt{\frac{nm}{c}}, W\frac{nm}{c}\right)\right)$ bit-meters/sec is achievable in a $(1, c)$-network, under the channel model 1. The result is then extended to a (m, c)-network by using Lemma 2.

Step 1: We consider a torus of unit area. Let $k = \min\left(c, \frac{n}{8}\right)$. This implies that $k \leq c$. Partition the square area into $\frac{n}{8k}$ equal-sized square cells, and place $8k$ nodes in each cell. Since the total area is 1, each cell has an area of $\frac{8k}{n}$, and sides of length $l = \sqrt{\frac{8k}{n}}$.

Step 2: The $8k$ nodes within each cell are distributed by placing k nodes at each of the eight positions shown in Figure 3.4. Nodes placed at locations S1, S2, S3, S4 act as senders, and nodes placed at remaining locations act as receivers. The sender locations S1 through S4 are at a distance of $r\Delta$ from the center of the cell (recall that Δ is the "guard" parameter from the protocol model of interference), where $r = \frac{l}{2(1+2\Delta)} = \frac{1}{(1+2\Delta)}\sqrt{\frac{2k}{n}}$. The receiver locations R1 through R4 are at a distance of $r(1 + \Delta)$ from the center of the cell. Therefore, the distance between S1-R1, S2-R2, S3-R3, and S4-R4 is equal to r. Each receiver location is at a distance of $r\Delta$ from nearest edge of the cell, and each sender location is at a distance of $r(1 + \Delta)$ from the nearest edge of the cell.

Step 3: Label the k nodes in any location (S1 through S4, R1 through R4) as 1 through k. The j^{th} node in each sender location, $1 \leq j \leq k$, communicates with the j^{th} node in the nearest receiver location (at a distance of r) on channel j. Consider any pair of communicating nodes A and B that are located at, say, S1 and R1 respectively. Then, the nearest senders within the cell, other

than A (located at S1), which are sending on the same channel as A are located at one of S2, S3, S4, and are at least a distance of $r(1+\Delta)$ away from B (located at R1). Similarly, in every cell, senders are at least $r(1+\Delta)$ distance from the cell boundary. Therefore, senders in adjacent cells of B are at least a distance of $r(1+\Delta)$ away from B as well. Hence, under the protocol model of interference, the transmission between A and B is not interfered with by any other transmission in the network, and this property holds for all communicating pairs.

From the above construction, there are $\frac{n}{2}$ pairs of nodes in the $(1,c)$-network, each transmitting at a rate of $\frac{W}{c}$ over a distance $r = \frac{1}{(1+2\Delta)}\sqrt{\frac{2k}{n}}$. Hence, the total capacity of the network (summing over all n nodes) is $\frac{n}{2}\frac{W}{c}r = \frac{W}{c}\frac{1}{(1+2\Delta)}\sqrt{\frac{nk}{2}}$ bit-meters/sec. Recall that $k = \min(c, \frac{n}{8})$. Substituting for k in the above derivation, we obtain the capacity of a $(1,c)$-network to be $\Omega\left(MIN_O\left(W\sqrt{\frac{n}{c}}, W\frac{n}{c}\right)\right)$ bit-meters/sec under channel model 1, since Δ is a constant.

Using Lemma 2, the capacity of a (m,c)-network, under the arbitrary network setting and channel model 1, is $\Omega\left(MIN_O\left(W\sqrt{\frac{n}{\lfloor\frac{c}{m}\rfloor}}, W\frac{n}{\lfloor\frac{c}{m}\rfloor}\right)\right)$ bit-meters/sec. Since $\frac{1}{\lfloor\frac{c}{m}\rfloor} \geq \frac{1}{\frac{c}{m}}$, we have the capacity of arbitrary networks to be $\Omega\left(MIN_O\left(W\sqrt{\frac{mn}{c}}, W\frac{mn}{c}\right)\right)$ bit-meters/sec, which leads to the following theorem:

Theorem 4. *The achievable network capacity of a (m,c)-arbitrary network under channel model 1 is as follows:*

1. *When $\frac{c}{m}$ is $O(n)$, network capacity is $\Omega\left(W\sqrt{\frac{nm}{c}}\right)$ bit-meters/sec.*
2. *When $\frac{c}{m}$ is $\Omega(n)$, network capacity is $\Omega\left(W\frac{nm}{c}\right)$ bit-meters/sec.*

The result for channel model 2 can be similarly derived, and is given by:

Theorem 5. *The achievable network capacity of a (m,c)-arbitrary network under channel model 2 is as follows:*

1. *When $\frac{c}{m}$ is $O(n)$, network capacity is $\Omega\left(W\sqrt{nmc}\right)$ bit-meters/sec.*
2. *When $\frac{c}{m}$ is $\Omega(n)$, network capacity is $\Omega\left(Wnm\right)$ bit-meters/sec.*

The upper bound and lower bound of the capacity of arbitrary networks have the same order, indicating that the bounds are tight.

3.3.3 Implications

A common scenario is when the number of channels is not too large ($\frac{c}{m} = O(n)$). Under this scenario, the capacity of a (m, c)-network in the arbitrary setting scales as $\Theta\left(W\sqrt{\frac{nm}{c}}\right)$ under channel model 1. Similarly, under channel model 2, the capacity of the network scales as $\Theta\left(W\sqrt{nmc}\right)$. Under either model, the capacity of a (m, c)-network goes down by a factor of $1-\sqrt{\frac{m}{c}}$, when compared with a (c, c)-network. Therefore, doubling the number of interfaces at each node (as long as number of interfaces is smaller than the number of channels) increases the channel capacity by a factor of only $\sqrt{2}$. Furthermore, the ratio between m and c demarcates the capacity regions, rather than the individual values of m and c. Increasing the number of interfaces may result in a linear increase in the cost but only a sub-linear (proportional to square-root of number of interfaces) increase in the capacity. Therefore, the optimal number of interfaces to use may be smaller than the number of channels depending on the relationship between cost of interfaces and utility obtained by higher capacity.

Different network architectures have been proposed for utilizing multiple channels when the number of available interfaces is smaller than the number of available channels [4, 35, 36]. The construction used in proving lower bound shows that capacity is maximized when all channels are utilized. One architecture used in the past [4] is to use only m channels when m interfaces are available, leading to wastage of the remaining $c - m$ channels. That architecture results in a factor of $1 - \frac{m}{c}$ loss in capacity which can be significantly higher than the optimal $1 - \sqrt{\frac{m}{c}}$ loss (when $\frac{c}{m} = O(n)$). Hence, in general, higher capacity may be achievable by *architectures that use all channels*, possibly by dynamically switching channels.

3.4 Capacity results for random networks

We assume that n nodes are randomly located on the surface of a torus of unit area. Each node selects a destination uniformly at random from the remaining nodes, and sends $\lambda(n)$ bits/sec to the destination. The highest value of $\lambda(n)$ which can be supported by *every* source-destination pair with high probability is defined as the *per-node throughput* of the network. The traffic between a

source-destination pair is referred to as a "flow". Since there are a total of n flows, the network capacity is defined to be $n\lambda(n)$.

Note that each node picks a destination node randomly, and therefore, a node may be the destination of multiple flows. Let $D(n)$ be the maximum number of flows for which a node in the network is a destination. We use the following result to bound $D(n)$.

Lemma 3. *The maximum number of flows for which a node in the network is a destination, $D(n)$, is $\Theta\left(\frac{\log n}{\log\log n}\right)$, with high probability.*

Proof. The process of nodes selecting a random destination may be mapped to the well-known "Balls into Bins" problem [37]. Each source node may be viewed as a "ball", and each destination node may be viewed as a "bin". The process of selecting a destination node may be viewed as randomly dropping a "ball" into a "bin". Based on this mapping, the proof of the lemma follows from well-known results (cf. [37], Section 4). □

3.4.1 Upper bound for random networks

The capacity of multichannel random networks is limited by three constraints, and each of them is used to obtain a bound on the network capacity. The minimum of the three bounds (the bounds depend on ratio between the number of channels c and the number of interfaces m) is an upper bound on the network capacity. While there may be other constraints on capacity as well, the constraints we consider are sufficient to provide a tight bound. We derive the bounds under channel model 1, but the results are applicable under channel model 2 as well.

Constraint 1 – Connectivity constraint: The capacity of random networks is constrained by the need to ensure that the network is connected, so that every source-destination pair can successfully communicate. Since node locations are randomly chosen, there is some minimum transmission range each node should use to ensure that the network is connected. Since all transmissions cover at least an area proportional to the square of the minimum transmission range, there is a bound on the number of simultaneous transmissions that can occur in the network. Based on this observation, Gupta and Kumar [14] have presented one bound on the network capacity to be $O\left(W\sqrt{\frac{n}{\log n}}\right)$ bits/sec. This bound is applicable to multichannel networks as well.

Constraint 2 – Interference constraint: A random network is a special case of an arbitrary network, and therefore the arbitrary network constraints are applicable to random networks as well. Therefore, the capacity of multichannel random networks is also constrained by interference (this is same as the constraint 1 listed for arbitrary networks in Section 3.3.1). This constraint was already captured in the upper bound for arbitrary networks, and we had obtained a bound of $O\left(W\sqrt{\frac{nm}{c}}\right)$ bit-meters/sec. In a random network, each of the n source-destination pairs are separated by an average distance of $\Theta(1)$ meter. Consequently, the network capacity of random networks is at most $O\left(W\sqrt{\frac{nm}{c}}\right)$ bits/sec. We do not explicitly use the second arbitrary network constraint ("Interface bottleneck constraint" from Section 3.3.1) in the random network proof as the bounds established by that constraint are not tight, and that bound is subsumed by the bound for "destination bottleneck constraint" below.

Constraint 3 – Destination bottleneck constraint: The capacity of a multichannel network is constrained by the data that can be received by a destination node. Consider a node X which is the destination of the maximum number (that is, $D(n)$) of flows. Recall that in a (m, c)-network, each channel supports a data rate of $\frac{W}{c}$ bits/sec. Therefore, the total data rate at which X can receive data over m interfaces is $\frac{Wm}{c}$ bits/sec. Since X has $D(n)$ incoming flows, the data rate of the minimum rate flow is at most $\frac{Wm}{cD(n)}$ bits/sec. Therefore, by definition of $\lambda(n)$, $\lambda(n) \leq \frac{Wm}{cD(n)}$, implying that network capacity (which by definition is $n\lambda(n)$) is at most $O\left(\frac{Wmn}{cD(n)}\right)$ bits/sec. Substituting for $D(n)$ from Lemma 3, the network capacity is at most $O\left(\frac{Wmn\log\log n}{c\log n}\right)$ bits/sec.

The bound obtained from constraint 3 is applicable to any network, including mobile networks, as long as the destination of every flow is randomly chosen among the nodes in the network. Even when $m = c$, this bound implies that the per-flow throughput, $\lambda(n)$, is at most $O\left(\frac{W\log\log n}{\log n}\right)$ bits/sec. Previous results on capacity of mobile networks [15,17,38] have stated a per-flow throughput of $O(W)$ bits/sec is possible, as in their models, each node does not randomly select a destination node. In our work, we choose the destination of a flow randomly from among $n-1$ possible destinations, similar to Gupta and Kumar [14]. Considering our discussion above, the $O(W)$ bits/sec bound with mobility cannot apply when destination nodes are randomly chosen. The previous results for mobile networks hold under other models of selecting destination nodes, wherein each

node is the destination of at most $O(1)$ flows (for example, such a constraint is satisfied when permutation routing is used).

Combining the above three bounds, the capacity of a random network, under channel model 1, is upper bounded by $O\left(MIN_O\left(W\sqrt{\frac{n}{\log n}}, W\sqrt{\frac{nm}{c}}, \frac{Wmn\log\log n}{c\log n}\right)\right)$ bits/sec. From this, we have the following theorem on the upper bound on capacity of random networks (Figure 3.2 has a pictorial representation).

Theorem 6. *The upper bound on the capacity of a (m, c)-random network under channel model* 1 *is as follows:*

1. *When $\frac{c}{m}$ is $O(\log n)$, network capacity is $O\left(W\sqrt{\frac{n}{\log n}}\right)$ bits/sec.*
2. *When $\frac{c}{m}$ is $\Omega(\log n)$ and also $O\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, network capacity is $O\left(W\sqrt{\frac{nm}{c}}\right)$ bits/sec.*
3. *When $\frac{c}{m}$ is $\Omega\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, the network capacity is $O\left(\frac{Wmn\log\log n}{c\log n}\right)$ bits/sec.*

The result for channel model 2 can be similarly derived, and is given by:

Theorem 7. *The upper bound on the capacity of a (m, c)-random network under channel model* 2 *is as follows:*

1. *When $\frac{c}{m}$ is $O(\log n)$, network capacity is $O\left(Wc\sqrt{\frac{n}{\log n}}\right)$ bits/sec.*
2. *When $\frac{c}{m}$ is $\Omega(\log n)$ and also $O\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, network capacity is $O\left(W\sqrt{nmc}\right)$ bits/sec.*
3. *When $\frac{c}{m}$ is $\Omega\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, the network capacity is $O\left(\frac{Wmn\log\log n}{\log n}\right)$ bits/sec.*

An interesting observation from the upper bound result is that as long as $\frac{c}{m}$ is $O(\log n)$, the number of interfaces has no impact on channel capacity. This implies that when the number of channels is $O(\log n)$ (which is the common case today), there is no loss in network capacity even if each node has a single interface.

3.4.2 Constructive lower bound

The lower bound is established by constructing a routing scheme and a transmission schedule for any random network. The lower bound matches the upper bound implying that the bounds are

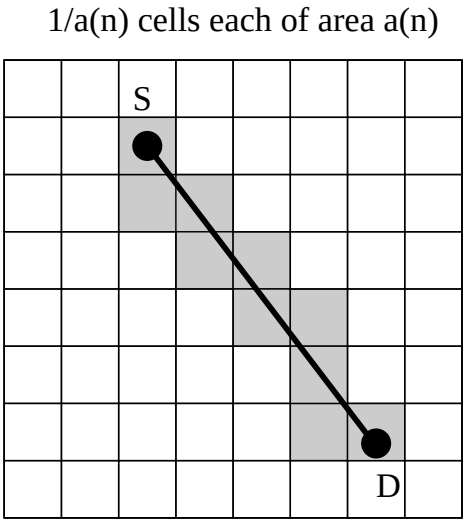

Figure 3.5: The unit area is divided into square cells. Packets are routed through the cells intersected by the line joining the source and the destination. Within each cell, a specific node is chosen for forwarding all packets of a flow.

tight. We will provide a construction for a $(1, c)$-network (a network wherein each node has a single interface) under channel model 1, and then invoke Lemma 2 to extend the result to a (m, c)-network. The steps involved in the construction are described next.

Cell construction

The surface of the unit torus is divided using a square grid into square cells (see Figure 3.5), each of area $a(n)$, similar to the approach used in [17]. The key difference in our work from [17] is that the size of the cell, $a(n)$, varies with the number of channels, and has to be carefully chosen to meet multiple constraints (which are described later in the text). In particular, we set $a(n) = \min\left(\max\left(\frac{100\log n}{n}, \frac{c}{n}\right), \left(\frac{1}{D(n)}\right)^2\right)$, where $D(n) = \Theta\left(\frac{\log n}{\log\log n}\right)$ as described before. Intuitively, the three values that influence $a(n)$ are based on the three constraints that were described in the upper bound proof: cell size needed to ensure connectivity, cell size needed when capacity is constrained by interference, and cell size needed when capacity is constrained by the maximum number of flows to any destination node, respectively.

We need to bound the number of nodes that are present in each cell, which is derived in Lemma 4.

Lemma 4. *If $a(n) \geq \frac{100 \log n}{n}$, then each cell has $\Theta(na(n))$ nodes per cell, with high probability.*

Proof. A similar result was stated in [17] without proof. Here we provide a proof based on VC-theory (see [39] for details on VC-theory), similar to the approach used by Gupta and Kumar [14]. The total number of square cells is $\frac{1}{a(n)}$. Since nodes are randomly located on the torus, the probability that any given node will lie in a specific cell is $a(n)$. We want to derive bounds on number of nodes in *every* cell in the square grid, which requires a proof of uniform convergence. The set of axis-parallel squares $\mathcal{C}$ are known to have VC-dimension 3. By applying the Vapnik-Chervonekis theorem [40], similar to the approach used in [14], we have the following bound on the number of nodes N_C in any cell C:

$$\text{Prob}\left(\sup_{C \in \mathcal{C}} \left| \frac{N_C}{n} - a(n) \right| \leq \frac{50 \log n}{n}\right) > 1 - \frac{50 \log n}{n} \tag{3.8}$$

where the constants in the above expression have been carefully chosen to satisfy the Vapnik-Chervonekis theorem. The above result implies that with high probability, we have

$$na(n) - 50 \log n \leq N_C \leq na(n) + 50 \log n$$

provided that $a(n) \geq \frac{100 \log n}{n}$.

Hence, we can conclude that the number of nodes in any cell is $\Theta(na(n))$ with high probability, as long as $a(n) \geq \frac{100 \log n}{n}$. □

By construction, we ensure that $a(n) \geq \frac{100 \log n}{n}$ for large n (because $\max\left(\frac{100 \log n}{n}, \frac{c}{n}\right)$ is at least $\frac{100 \log n}{n}$, and $\left(\frac{1}{D(n)}\right)^2$ is asymptotically larger than $\frac{100 \log n}{n}$). Thus, with our choice of $a(n)$, Lemma 4 holds for suitably large n, and each cell has $\Theta(na(n))$ nodes per cell, *whp*.

The transmission range[2] of each node, $r(n)$, is set to be $\sqrt{8a(n)}$. With this transmission range, a node in one cell can communicate with any node in its eight neighboring cells. Note that when the cell size $a(n)$ increases, larger transmission range is required, as $r(n)$ is dependent on $a(n)$.

[2]Transmission range is defined to be the maximum distance over which any node can communicate.

A transmission originating from a node S interferes with another transmission from A destined to B, only if S is within a distance of $(1+\Delta)r(n)$ of receiver B (using the interference definition of protocol model). Since the distance between A and B is at most $r(n)$, the distance between the two transmitters, S and A, must be less than $(2+\Delta)r(n)$ if the transmissions were to interfere. Hence, any transmission can possibly interfere with only those transmissions from transmitters within a distance of $(2+\Delta)r(n)$. Therefore, nodes in a cell can be interfered with by only nodes in cells within a distance of $(2+\Delta)r(n)$, and this interfering area can be completely enclosed in a larger square of side $3(2+\Delta)r(n)$ (this is a loose bound). Consequently, there are at most $\frac{(3(2+\Delta)r(n))^2}{a(n)} = 72(2+\Delta)^2$ interfering cells (recall $r(n) = \sqrt{8a(n)}$). Hence, the number of interfering cells, $k_{inter} \leq 72(2+\Delta)^2$, is a constant that only depends on Δ (and is independent of $a(n)$ and n).

Routing Scheme

Packets are routed through the cells that lie along the straight line joining the source and the destination node. A node in each cell through which the line passes is used to relay traffic along that flow (we will describe the choice of the node later). Figure 3.5 shows an example of the cells used to route data for a flow between source S and destination D.

In previously proposed constructions for proving lower bound on capacity [14,17], it was immaterial which node in a chosen cell forwarded packets for some flow. However, such an approach may "*overload*" certain nodes, leading to capacity degradation, when the number of interfaces per node is smaller than the number of channels. Consequently, it is important to ensure that the routing load is distributed among the nodes in a cell. This is a key extension to the routing procedure used in earlier capacity results [14], and the extension is described next.

For each flow passing through a cell, one node in the cell is "assigned" to the flow. The assigned node of a flow in a cell is the only node in that cell which may receive/transmit data along that flow. The assignment is done using a *flow distribution procedure* as below:

Step 1 – Assign source and destination nodes: For any flow that originates in a cell, the source node S is assigned to the flow (S is necessarily in the originating cell). Similarly, for any flow that terminates in a cell, the destination node D is assigned to the flow. Since a single node in each

cell is allowed to receive or transmit data for a flow, it is required that the source and destination nodes be assigned to flows originating or terminating from them.

Step 2 – Balance distribution of remaining flows: After step 1 is complete, we are left with only those flows that pass through a cell. Each such remaining flow passing through a cell is assigned to the node in the cell that has the least number of flows assigned to it so far. This step balances the assignment of flows to ensure that all nodes are assigned (nearly) the same number of flows. The node assigned to a flow will receive packets from some node in the previous cell and send the packet to a node in the next cell.

Each node is the originator of one flow. Each node is the destination of at most $D(n)$ flows, which by Lemma 3 is $\Theta\left(\frac{\log n}{\log \log n}\right)$. Therefore, step 1 of the flow distribution procedure assigns to each node at most $1 + D(n)$ flows.

We use the following lemma to bound the number of source-destination lines that pass through any cell (and are assigned in step 2); we omit the proof as it has already been presented earlier in [17].

Lemma 5. *The maximum number of source-destination lines that intersect any cell (including lines originating and terminating in the cell) is* $O\left(n\sqrt{a(n)}\right)$*, with high probability.*

Step 2 of the flow distribution procedure carefully assigns the remaining flows among the nodes in the cell to ensure that all nodes end up with nearly same number of flows. By Lemma 4, each cell has $\Theta\left(na(n)\right)$ nodes, and by Lemma 5 at most $O\left(n\sqrt{a(n)}\right)$ flows pass through a cell. Therefore, step 2 will assign to any node in the network at most $O\left(\frac{1}{\sqrt{a(n)}}\right)$ flows. Therefore the total flows assigned to any node is at most $O\left(1 + D(n) + \frac{1}{\sqrt{a(n)}}\right)$. Based on the rules to set $a(n)$, described earlier, the maximum value of $a(n)$ is at most $\left(\frac{1}{D(n)}\right)^2$, which implies $\frac{1}{\sqrt{a(n)}}$ is at least $D(n)$. Hence, the total flows assigned to any node is always asymptotically dominated by $\frac{1}{\sqrt{a(n)}}$, and is therefore equal to $O\left(\frac{1}{\sqrt{a(n)}}\right)$ flows.

Scheduling transmissions

The transmission scheduling scheme is responsible for generating a transmission schedule for each node in the $(1, c)$-network that satisfies the following constraints:

Constraint 1: When a node X transmits a packet to a node Y over a channel j for some flow, X and Y should not be scheduled to transmit/receive at the same time for any other flow (since each node is assumed to have a single interface in the construction).

Constraint 2: Any two simultaneous transmissions on any channel should not interfere.

The multichannel construction differs from the mechanisms used in earlier constructions [14,17] in two ways. First, the scheduling is on a per-node basis since flows are distributed among nodes, whereas in the past work it was sufficient to schedule on a per-cell basis. Second, since there is a single interface, but c channels are available (recall that we are assuming a $(1, c)$-network for now), the schedule has to additionally ensure that at most a single transmission/reception is scheduled for a node at any time (constraint 1).

We build a suitable schedule using a two-step process. In the first step, we satisfy constraint 1 by scheduling transmissions in "edge-color" slots so that at every node during any edge-color slot, at most one transmission or reception is scheduled. In the second step, we satisfy constraint 2 by dividing each edge-color slot into "mini-slots", and assigning mini-slots to channels such that any scheduled transmission is interference-free. By using the two-step process, each transmission in a mini-slot satisfies both constraint 1 and constraint 2.

Step 1 – Build a routing graph: We build a graph, called the "routing graph", whose vertices are the nodes in the network. One edge is inserted between all node pairs, say A and B, for every flow on which A and B are consecutive nodes (the routing scheme for selecting nodes along a flow was described earlier). Therefore, by this construction, every hop[3] in the network along any flow is associated with one edge in the routing graph. The resulting routing graph is a multi-graph[4] in which each node has at most $O\left(\frac{1}{\sqrt{a(n)}}\right)$ edges, since each flow through a node can result in at most two edges, one incoming and one outgoing, and we have already shown that each node is assigned to at most $O\left(\frac{1}{\sqrt{a(n)}}\right)$ flows. It is a well-known result [41] that a multi-graph with at most e edges per vertex can be edge-colored[5] with at most $\frac{3e}{2}$ colors. Therefore, the routing graph can be edge colored with at most some $f = O\left(\frac{1}{\sqrt{a(n)}}\right)$ colors.

[3]A hop is a pair of consecutive nodes on a flow.

[4]A graph with possibly multiple edges between a pair of nodes.

[5]Edge-coloring requires any two edges incident on a common vertex to use different colors.

We use edge coloring to ensure that when a transmission is scheduled along an edge, the interfaces on the nodes at either end of the edge are free, thereby satisfying constraint 1. We divide every 1 second period into f (which is $O\left(\frac{1}{\sqrt{a(n)}}\right)$) "edge-color" slots, each of length $\frac{1}{f}$ (which is $\Omega\left(\sqrt{a(n)}\right)$) seconds. Each of these edge-color slots is associated with an unique edge color. An edge is scheduled for transmission in the slot associated with its edge color. Since edge coloring ensures that at a vertex, all edges connected to the vertex use different colors, each node will have at most one transmission/reception scheduled in any edge-color slot. By construction, each edge corresponds to a hop in the network. Therefore this scheme ensures that during every 1 second interval, along any flow in the network, one transmission is scheduled on each hop of a flow.

Step 2 – Build an interference graph: In step 2, each edge-color slot is further sub-divided into "mini-slots" as explained below, and every node has an opportunity to transmit in some mini-slot. We develop a schedule for using mini-slots, which satisfies constraint 2. The schedule decides on which mini-slot within an edge-color slot and on what channel a node may transmit, and the same schedule is used in every edge-color slot.

We build another graph, called the "interference graph", wherein, vertices are nodes in the network, and there is an edge between two nodes if they may interfere with each other. Since every cell has at most some constant k_{inter} number of cells that may interfere with each other, and each cell has $\Theta(na(n))$ nodes, each node has at most $g = O(na(n))$ edges in the interference graph. It is well-known that a graph with maximum degree e can be vertex-colored[6] with at most $e+1$ colors [41]. Therefore, the graph can be vertex-colored with some $O(na(n))$ colors, i.e., at most $k_1 na(n)$ colors for some constant k_1. Transmissions by two nodes assigned the same vertex-color do not interfere with each other. Hence, they can be scheduled to transmit on the same channel at the same time. On the other hand, nodes colored with different colors may interfere with each other, and need to be scheduled either on different channels, or at different time slots on the same channel.

We divide each edge-color slot into $\left\lceil \frac{k_1 na(n)}{c} \right\rceil$ mini-slots on every channel, and number the slots on each channel from 1 to $\left\lceil \frac{k_1 na(n)}{c} \right\rceil$. There is a total of $c\left\lceil \frac{k_1 na(n)}{c} \right\rceil$ mini-slots across the c channels. Channels are numbered from 1 to c. A node which is allocated a color $p, 1 \leq p \leq k_1 na(n)$ is

[6]Vertex-coloring requires any two vertices sharing a common edge to use different colors.

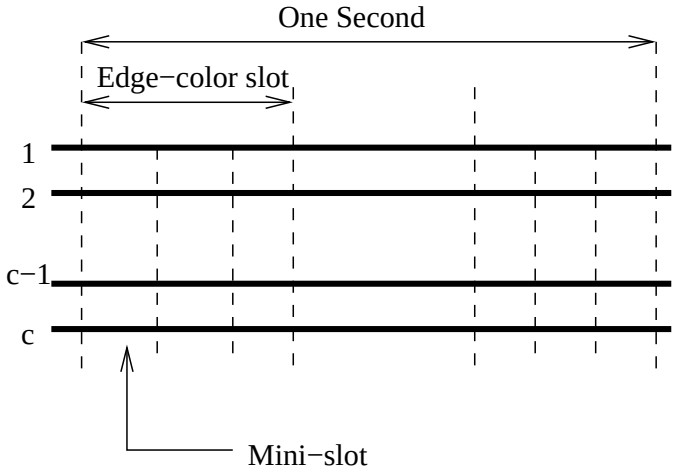

Figure 3.6: Transmission schedule: Every hop along every flow is assigned to exactly one edge-color slot in each one second interval. Within the edge-color slot assigned to a hop, a specific mini-slot is chosen during which the transmitter node on that hop may transmit.

allowed to transmit in mini-slot $\lceil \frac{p}{c} \rceil$ on channel $(p \bmod c) + 1$. The node may actually transmit if the edge-coloring has allocated an outgoing edge from the node to the corresponding edge-color slot.

Figure 3.6 depicts a schedule of transmissions on the network developed after the two-step scheduling process. The first step allocates one edge-color slot for each hop of every flow. The second step decides within each edge-color slot when the transmitter node on a hop may actually transmit a packet.

From step 1, each edge-color slot is of length $\Omega\left(\sqrt{a(n)}\right)$ seconds. From step 2, each edge-color slot is sub-divided into $\left\lceil \frac{k_1 na(n)}{c} \right\rceil$ mini-slots. Therefore, each mini-slot is of length $\Omega\left(\frac{\sqrt{a(n)}}{\left\lceil \frac{k_1 na(n)}{c} \right\rceil}\right)$ seconds. Each channel can transmit at the rate of $\frac{W}{c}$ bits/second. Hence, in each mini-slot, $\lambda(n) = \Omega\left(\frac{W\sqrt{a(n)}}{c\left\lceil \frac{k_1 na(n)}{c} \right\rceil}\right)$ bits can be transported. Since $\left\lceil \frac{k_1 na(n)}{c} \right\rceil \leq \frac{k_1 na(n)}{c} + 1$, we have, $\lambda(n) = \Omega\left(\frac{W\sqrt{a(n)}}{k_1 na(n)+c}\right)$ bits/sec. Depending on the asymptotic order of c, either $na(n)$ or c will dominate the denominator of $\lambda(n)$. Hence, $\lambda(n) = \Omega\left(MIN_O\left(\frac{W}{n\sqrt{a(n)}}, \frac{W\sqrt{a(n)}}{c}\right)\right)$ bits/sec. Since each flow is scheduled to receive one mini-slot on each hop during every 1 second interval, every source-destination flow can support a per-node throughput of $\lambda(n)$ bits/sec. Therefore, the total network

capacity is equal to $n\lambda(n)$ which is equal to $\Omega\left(MIN_O\left(\frac{W}{\sqrt{a(n)}}, \frac{Wn\sqrt{a(n)}}{c}\right)\right)$ bits/sec.

Recall that $a(n)$ is set to $\min\left(\max\left(\frac{100\log n}{n}, \frac{c}{n}\right), \left(\frac{1}{D(n)}\right)^2\right)$, where $D(n) = \Theta\left(\frac{\log n}{\log\log n}\right)$. Substituting for $a(n)$ (the three possible values of $a(n)$ gives rise to three capacity regions) in the equation for capacity (derived above), we have the result:

Theorem 8. *The achievable capacity of a $(1, c)$-random network under channel model 1 is as follows:*

1. *When c is $O(\log n)$, $a(n) = \Theta\left(\frac{\log n}{n}\right)$, and the network capacity is $\Omega\left(W\sqrt{\frac{n}{\log n}}\right)$ bits/sec.*
2. *When c is $\Omega(\log n)$ and also $O\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, $a(n) = \Theta\left(\frac{c}{n}\right)$, and the network capacity is $\Omega\left(W\sqrt{\frac{n}{c}}\right)$ bits/sec.*
3. *When c is $\Omega\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, $a(n) = \Theta\left(\left(\frac{\log\log n}{\log n}\right)^2\right)$, and the network capacity is $\Omega\left(\frac{Wn\log\log n}{c\log n}\right)$ bits/sec.*

Using Lemma 2, the results for a (m, c)-network can be obtained by replacing every usage of c in Theorem 8 by $\frac{c}{m}$. Therefore, we have:

Theorem 9. *The achievable capacity of a (m, c)-random network under channel model 1 is as follows:*

1. *When $\frac{c}{m}$ is $O(\log n)$, $a(n) = \Theta\left(\frac{\log n}{n}\right)$, and the network capacity is $\Omega\left(W\sqrt{\frac{n}{\log n}}\right)$ bits/sec.*
2. *When $\frac{c}{m}$ is $\Omega(\log n)$ and also $O\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, $a(n) = \Theta\left(\frac{c}{mn}\right)$, and the network capacity is $\Omega\left(W\sqrt{\frac{nm}{c}}\right)$ bits/sec.*
3. *When $\frac{c}{m}$ is $\Omega\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, $a(n) = \Theta\left(\left(\frac{\log\log n}{\log n}\right)^2\right)$, and the network capacity is $\Omega\left(\frac{Wmn\log\log n}{c\log n}\right)$ bits/sec.*

The result for channel model 2 can be similarly derived, and is given by:

Theorem 10. *The achievable capacity of a (m, c)-random network under channel model 2 is as follows:*

1. *When $\frac{c}{m}$ is $O(\log n)$, $a(n) = \Theta\left(\frac{\log n}{n}\right)$, and the network capacity is $\Omega\left(Wc\sqrt{\frac{n}{\log n}}\right)$ bits/sec.*

2. *When* $\frac{c}{m}$ *is* $\Omega(\log n)$ *and also* $O\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, $a(n) = \Theta\left(\frac{c}{mn}\right)$, *and the network capacity is* $\Omega\left(W\sqrt{nmc}\right)$ *bits/sec.*

3. *When* $\frac{c}{m}$ *is* $\Omega\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, $a(n) = \Theta\left(\left(\frac{\log\log n}{\log n}\right)^2\right)$, *and the network capacity is* $\Omega\left(\frac{Wmn\log\log n}{\log n}\right)$ *bits/sec.*

The lower bound matches the upper bound implying that the bounds are tight. Recall that the transmission range $r(n)$ has been set to $\sqrt{8a(n)}$. Hence, the *transmission range is larger* in case 2 and case 3 of Theorem 9 as compared to case 1 (since $a(n)$ increases). This implies that in multichannel networks with large number of channels, higher transmission power is necessary for meeting capacity bounds than is required in a single channel network.

3.4.3 Implications

Figure 3.7 plots the network capacity as the number of channels in the network is scaled, for a one interface and a two interface network. The figure plots the scaling with some fixed n, without accounting for constants, for the two channel models. As we can see from the figure, under the channel model 1, the total bandwidth is fixed, and the network capacity reduces when the number of channels increases. In contrast, under channel model 2, bandwidth is added when the number of channels increases, thereby increasing network capacity (up to a point). Note that the results under the two models are not contradictory, because the capacity always degrades with more channels when compared to the capacity in a (c, c)-network. Furthermore, when the number of interfaces is increased, there is no improvement in capacity as long as $\frac{c}{m}$ is $O(\log n)$, but beyond that threshold, adding more interfaces improves capacity. Since the curves in Figure 3.7 plot the number of channels on the X-axis, the threshold $\frac{c}{m}$ is achieved for a larger value of c when m is increased.

Figure 3.8 plots the network capacity as the ratio of channels to interfaces, $\frac{c}{m}$, is increased for a one interface and a two interface network. When the number of interfaces is increased by some factor of k, then for the any given ratio of $\frac{c}{m}$, this implies that the number of channels is also increased by a factor of k. Since the total bandwidth is fixed under channel model 1, a k-fold increase in channels does not increase network capacity (and therefore, the curves for different number of interfaces overlap in Figure 3.8). However, since the bandwidth per channel is fixed

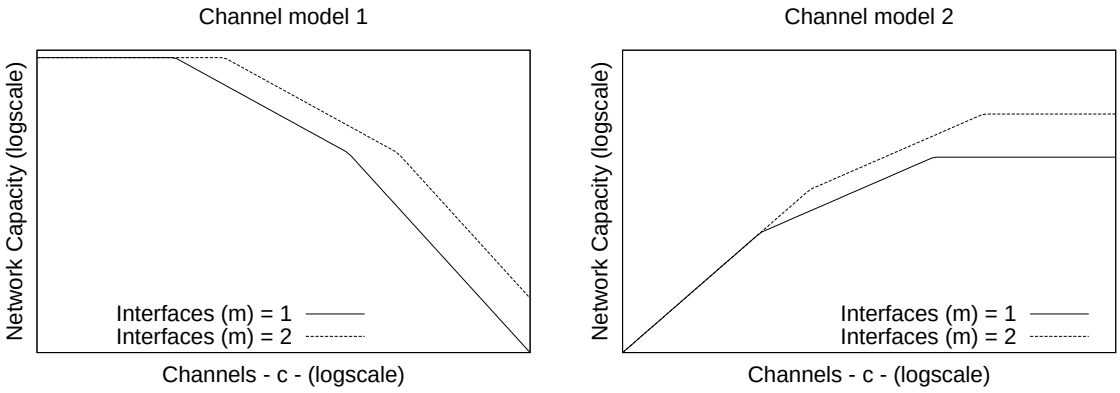

Figure 3.7: Example plot of network capacity as the number of channels is scaled. Capacity values are normalized to capacity in a $(1, 1)$-network.

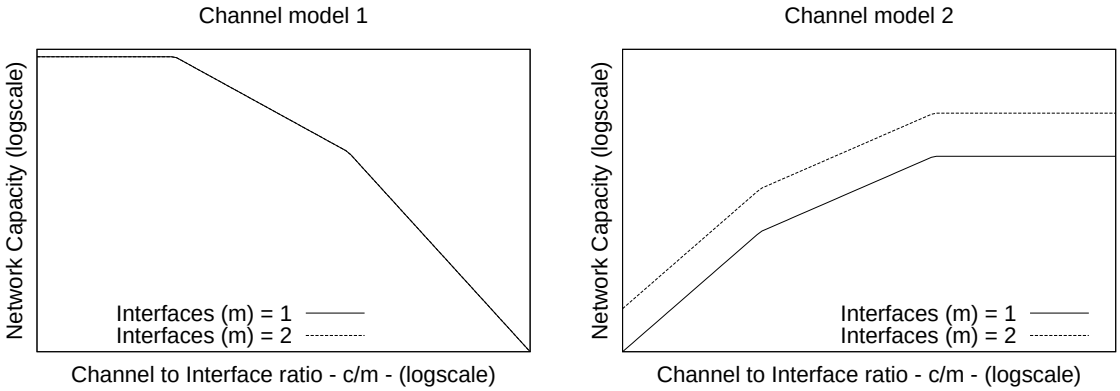

Figure 3.8: Example plot of network capacity as the ratio between channels to interfaces is scaled.Capacity values are normalized to capacity in a $(1, 1)$-network. The two curves under channel model 1 overlap.

under channel model 2, a k-fold increase in channels implies a k-fold increase in bandwidth, leading to a k-fold increase in the network capacity. In addition, under both channel models, the boundaries of the three capacity regions depend on $\frac{c}{m}$.

The results imply that the capacity of multichannel random networks with total channel data rate of W is the same as that of a single channel network with data rate W as long as the ratio $\frac{c}{m}$ is $O(\log n)$. When the number of nodes n in the network increases, we can also scale the number of channels (for example, by using additional bandwidth, or by dividing available bandwidth into multiple sub-channels). Even then, as long as the channels are scaled at a rate not more than

$\log n$, there is no loss in capacity even if a single interface is available at each node. In particular, if the number of channels c is a fixed constant, independent of the node density, then as the node density increases beyond some threshold density (at which point $c = O(\log n)$), there is no loss in capacity even if just a single interface is available per node. Thus, this result may be used to roughly estimate the number of interfaces each node has to be equipped with for a given node density and a given number of channels.

In a single channel random network, i.e., a $(1, 1)$-network, the capacity bottleneck arises out of the channel becoming fully utilized, and not because interface at any node is fully utilized. On an average, the interface of a node in a single channel network is busy only for $\frac{1}{X}$ fraction of the time, where X is the average number of nodes that interfere with a given node. In a $(1, 1)$-random network with n nodes, each node on an average has $\Theta(\log n)$ neighbors to maintain connectivity [14]. This implies that in a single channel network, each interface is busy for only $\Theta\left(\frac{1}{\log n}\right)$ time. Our construction utilizes this slack time of interfaces to support up to $O(\log n)$ channels without loss in capacity. In general, the loss in capacity in a random network is a function of the number of channels and the number of nodes in a neighborhood[7].

In earlier capacity results [14, 17], the transmission range, and therefore the neighborhood size, is a function of only the node density. However, for multichannel networks, the transmission range has to be chosen based on ratio of channels to interfaces, in addition to the node density. For example, with a given node density, when the ratio of number of channels to number of interfaces is large (specifically, $\omega(\log n)$), the number of interfaces in a neighborhood will be smaller than the total number of channels. Therefore, even if all the interfaces are being used continuously, it is not possible to fully saturate the available channels. This can result in significant capacity degradation.

The capacity degradation can be reduced by increasing the size of a neighborhood, thereby ensuring that the number of interfaces in a neighborhood is equal to the number of channels. Therefore, the lower bound construction requires the cell size to be chosen such that the number of interfaces (or nodes, when each node has a single interface) in each neighborhood is greater than

[7]The neighborhood of a node consists of all other nodes that may interfere with it.

or equal to the number of channels. Hence, it turns out that the optimal strategy for maximizing capacity when number of channels is large is to sufficiently increase the cell size $a(n)$, which implies that a *larger transmission range* $r(n)$ *is needed* to allow communication with neighboring cells. However, there is still some capacity loss because larger transmission range (than that is needed for connectivity alone) lowers capacity by "consuming" more area. In summary, in a single channel random network, the transmission range is chosen to be large enough to ensure connectivity. However, in the case of multichannel networks, the transmission range has to be chosen such that it is sufficiently large to ensure that all channels are utilized, in addition to guaranteeing connectivity.

3.5 Impact of switching delay

The previous discussion on multichannel capacity has not considered the impact of interface switching delay. When the number of interfaces at each node is smaller than the number of channels, interfaces may have to be switched between channels. Switching an interface from one channel to another may incur a switching delay, say S. For example, existing IEEE 802.11-based wireless interfaces require [35] between few tens to hundreds of microseconds to switch from one channel to another. However, switching delay is independent of the number of nodes in the network.

We will show that if there are no end-to-end delay constraints, switching delay will not affect network capacity. For this, we use the end-to-end delay constraint definition from [17]. Each packet is assumed to have a size L, and L is scaled with respect to the throughput obtained for each end-to-end flow. If each flow can transport λ bits/sec, then each flow is assumed to send packets of size $L = \lambda$. In the lower bound construction provided before, if packet sizes are set to λ bits, each packet traverses at least one hop in one second. Therefore, the end-to-end delay of a flow will be bounded by the number of hops on the flow, when there is no interface switching latency. Let us assume that the minimum end-to-end delay in the absence of interface switching latency is D_{opt}. A reasonable delay constraint in the presence of switching latency is to require that the end-to-end delay is at most a small constant multiple of D_{opt}; otherwise applications may see a large increase in the end-to-end delay. This requirement may be equivalently translated to allow a maximum packet size of L.

3.5.1 Capacity in the absence of end-to-end delay constraints

In the case of arbitrary networks, capacity bounds are met without requiring interface switching at all (as was shown in the construction used for lower bound). Hence, switching delay will not impact the capacity of arbitrary networks, even if there is an end-to-end delay constraint. In the absence of any end-to-end delay constraints, we show next that the capacity of random networks is independent of switching delay (the construction is described next).

In the construction we use to establish lower bound for random networks, interfaces may have to be switched between channels (when receiving data). In the worst case, an interface may have to be switched between channels for every packet transmission. If there is no end-to-end delay constraint, then we propose a simple "guard slot" approach which ensures that capacity loss can be made arbitrarily small even in the presence of switching delay.

The "guard slot" approach is as follows. Suppose that each packet is L bits long. This implies that the length of each edge color slot is $T = \frac{Lc}{W}$ seconds (since each channel supports a data rate of $\frac{W}{c}$ bits/sec under channel model 1). One simple way of hiding the interface switching delay S is to insert a "guard" slot of duration S between two "edge-color" slots during which all channels are idle, to ensure that there is sufficient time for interface switching. With this approach, the network capacity will be only $\frac{T}{T+S}$ fraction of the capacity when there is no switching delay. However, the capacity reduction can be made arbitrarily small by sending extremely large packets ($L \gg \lambda$) resulting in $T \gg S$, leading to large end-to-end delay. Therefore, in the absence of end-to-end delay constraints, by using large data packets, the capacity degradation in random networks can be made arbitrarily small.

3.5.2 Capacity in the presence of end-to-end delay constraints

From prior discussions, even in the presence of delay constraints, the capacity of arbitrary networks is not affected by switching delay, since switching is not required to meet the capacity bounds. In the case of random networks as well, the upper bound proofs do not mandate interfaces to be switched, and therefore, even with switching delay, there may be no change in the capacity. However, so far

we have not addressed the question whether the capacity of random networks is independent of the switching delay when there are end-to-end delay constraints.

In the presence of end-to-end delay constraints, switching delay *does reduce* the achievable network capacity in the lower bound constructions proposed earlier. For example, considering the guard-slot approach described above, when there is a restriction on the maximum packet size, each edge-color slot is bounded by some length T, and the network capacity will be only $\frac{T}{T+S}$ of the capacity without switching delay, We next describe an approach that shows using additional interfaces at each node is *sufficient* in many scenarios to hide the switching delay, even with end-to-end delay constraints.

The new approach simulates a *virtual interface* having zero switching delay using multiple physical interfaces that each have a switching delay S. By this construction, the use of $v-1$ additional interfaces per node can hide the switching delay, i.e., a (v,c)-network using interfaces with switching delay S can achieve the same capacity and end-to-end delay bounds as a $(1,c)$-network using one interface with 0 switching delay. This construction suggests that multiple interfaces are *sufficient* to overcome the impact of switching delay, though multiple interfaces may not be *necessary.*

Lemma 6. *Suppose that the time required for packet transmission in a $(1,c)$-network is $T=\frac{Lc}{W}$, and suppose $v=\left\lceil\frac{S}{T}\right\rceil+1$. Then a (v,c)-network built with interfaces having switching delay S, can achieve the same capacity and end-to-end delay as a $(1,c)$-network built with interfaces having* 0 *switching delay.*

Proof. Let us assume that each node has $v=\left\lceil\frac{S}{T}\right\rceil+1$ interfaces, each having a switching delay S. We build a *virtual interface* with zero switching delay by using the v physical interfaces, as shown in Figure 3.9. We consider any time interval of length vT. We divide this time into v slots of length T, and only allow the i^{th} interface, $1\leq i\leq v$, to transmit/receive in slot i. Thus, each physical interface is used for transmission/reception in one slot, and is idle for the next $(v-1)$ slots of total duration $(v-1)T$ seconds. Since $v=\left\lceil\frac{S}{T}\right\rceil+1$, we have:

$$(v-1)T \quad = \quad \left\lceil\frac{S}{T}\right\rceil T$$

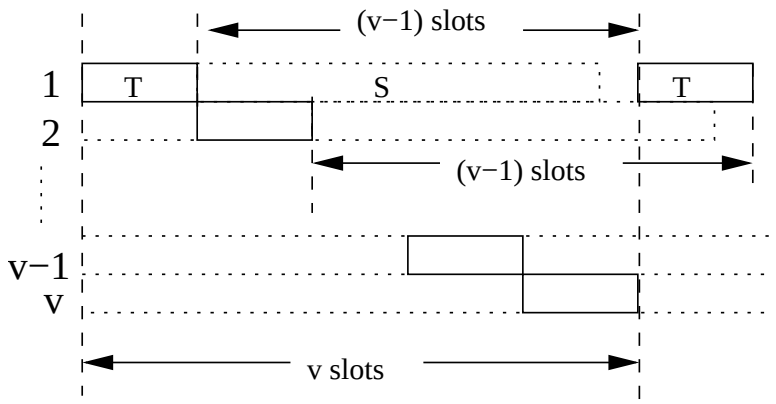

Figure 3.9: Constructing one virtual interface with zero switching delay by using v physical interfaces with switching delay S. Each packet transmission requires T seconds.

$$\geq \quad S$$

Hence, between two successive operations of a physical interface there is at least a gap of S, which ensures that switching delay is provisioned for. By this construction, the simulated virtual interface can continuously transmit/receive, with 0 switching delay. Therefore, a network using v interfaces having switching delay S, can mimic the behavior of a $(1, c)$-network built with interfaces having switching delay 0. □

From the previous lemma, by increasing the number of interfaces at each node by a factor of v, switching delay is completely hidden. We next discuss the capacity implications of using v physical interfaces at each node to construct a virtual interface, instead of directly using the v interfaces to send data in parallel.

From Theorem 9, we note that when the number of channels is $O(\log n)$ and there is no switching delay, the capacity of a (v, c)-network is the same as that of a $(1, c)$-network. Using this observation along with Lemma 6, we can conclude that by using the virtual interface technique, the capacity of a (v, c)-network with each interface having switching delay S is the same as the capacity of a (v, c)-network with each interface having switching delay 0. Hence, when the number of channels is $O(\log n)$, which is a scenario of significant practical interest, *there is no capacity loss even with switching delay, provided multiple interfaces are used.*

Again, from Theorem 9, we note that when the number of channels is larger ($\Omega(\log n)$) and there is no switching delay, the capacity of a $(1, c)$-network is *lower* than that of a (v, c)-network. Hence, using this observation along with Lemma 6, we can conclude that using the virtual interface technique when the number of channels is larger ($\Omega(\log n)$), a (v, c)-network with each interface having switching delay S *will have lower capacity* than a (v, c)-network with each interface having switching delay 0. Using Theorem 9, we can show that for this scenario, the capacity will be lower by a factor of $\frac{1}{\sqrt{v}} \approx \sqrt{\frac{T}{T+S}}$ (since $v \approx \frac{T+S}{T}$) when number of channels is between $\Omega(\log n)$ and $O\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$, and by a factor of $\frac{1}{v} \approx \frac{T}{T+S}$ when number of channels is $\Omega\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$. In contrast, if the guard slot approach is used, the capacity is lower by a factor $\frac{T}{T+S}$ in all cases, independent of the number of channels. Therefore, although there is a capacity loss with switching delay for certain scenarios using the virtual interface technique, it is still significantly better than the guard slot approach when the number of channels is small.

3.5.3 Other constructions

The constructions used to establish lower bound on capacity potentially require interface switches at several hops of a flow. Alternate constructions are possible [42] such that an interface switch is required on at most one hop of each flow. Such a construction may reduce the number of switches required in the network, and could be used to reduce the impact of switching delay.

The capacity analysis has assumed that each node has one flow active all the time. Typically, in deployed networks, every node may not have an active flow all the time. In such a scenario, nodes without active flows could still forward data for other nodes, and the interfaces of the inactive nodes may be viewed as "spare" interfaces that are available in the network. Such spare interfaces could also be used to hide interface switching delay, instead of requiring additional interfaces at each node to hide the switching delay. We defer a formal study on hiding interface switching delay by using spare interfaces to future work.

3.6 Capacity with fixed interfaces

In the previous section, we considered the capacity of multichannel networks when switching interfaces incurs a delay. In this section, we study the capacity of multichannel networks when interfaces do not switch at all. We assume that each interface is fixed to some channel, and the channel to which an interface is fixed can be set by the network designer. It may be beneficial to keep interfaces fixed when the interface switching delay is large. For tractability, we study the capacity problem under a slightly different model for selecting source-destination pairs, called the permutation traffic model. Under this model, we show that there is a degradation in capacity, proportional to the number of channels, when there is a single interface per node, and interfaces are not allowed to switch. However, the capacity degradation can be prevented if each node is equipped with two interfaces (and interfaces continue to be fixed on some channels).

A permutation is an one-to-one correspondence from a set $\{1, 2, .. n\}$ to itself. There are $n!$ possible permutations, and a random permutation is defined as a permutation chosen uniformly at random from all the possible permutations. The permutation routing model assumes that the source-destination pairs are chosen as a random permutation. This implies that each node is the source of exactly one flow, and each node is the destination of exactly one flow. In contrast, the traffic model used earlier in this chapter allowed nodes to be destinations of more than one flow. Permutation routing model has been assumed by other works on capacity in the past (e.g. [20, 21]), and is simpler to analyze. All the capacity constraints, except the destination bottleneck constraint, continue to hold under the permutation routing model. Because the destination bottleneck constraint is no longer applicable, region 3 of the capacity of random networks starts only when $\frac{c}{m} = \Omega(n)$. It can be easily shown that the random network capacity under permutation routing model is given by,

Theorem 11. *The capacity of a (m, c)-random network under channel model 1, and permutation traffic model, is as follows:*

1. *When $\frac{c}{m}$ is $O(\log n)$, the network capacity is $\Omega\left(W\sqrt{\frac{n}{\log n}}\right)$ bits/sec.*
2. *When $\frac{c}{m}$ is $\Omega(\log n)$ and also $O(n)$, the network capacity is $\Omega\left(W\sqrt{\frac{nm}{c}}\right)$ bits/sec.*

3. When $\frac{c}{m}$ is $\Omega(n)$, the network capacity is $\Omega\left(\frac{Wnm}{c}\right)$ bits/sec.

3.6.1 Capacity bound with a single fixed interface

When every node has a single fixed interface, nodes fixed on a certain channel cannot communicate with nodes fixed on any other channel. Network capacity depends on the smallest throughput obtained by any flow, and to ensure that the network capacity is greater than zero, any pair of source-destination nodes must be fixed to a common channel. This constraint precludes fixing nodes to channels arbitrarily.

Let us suppose that each node has been fixed on a channel, and this channel assignment has been done while ensuring that any source-destination pair uses the same channel. Let a channel $i, 1 \leq i \leq c$, have n_i nodes fixed on it, and let λ_i be the smallest flow throughput among the flows on channel i. Consider some channel i. Now, the n_i nodes must still satisfy the connectivity constraint to ensure that no nodes are disconnected from each other. As before, we assume that all nodes use a common transmission range. Therefore, the transmission range on channel i must be at least as large as the transmission range required when all nodes share a common channel, i.e., transmission range $r(n_i) = \Omega\left(\sqrt{\frac{\log n}{n}}\right)$. Furthermore, the average distance between the source-destination pairs on channel i continues to be $\Theta(1)$ meters. Using upper bound results from Gupta and Kumar [14], per-flow throughput of a network having n_i nodes using transmission range $r(n_i)$, and channel bandwidth $\frac{W}{c}$ (we assume channel model i) is upper-bounded as follows:

$$\lambda_i = O\left(\frac{W}{c}\frac{1}{n_i r(n_i)}\right) \tag{3.9}$$

Substituting for $r(n_i)$, we get

$$\lambda_i = O\left(\frac{W}{cn_i}\sqrt{\frac{n}{\log n}}\right)$$

By definition, the network-wide per-flow throughput, λ, is defined to be the minimum throughput achieved by any flow, i.e., $\lambda = \min_i(\lambda_i)$. Therefore, λ is limited by the per-flow capacity in the channel having the maximum number of nodes. Using this observation, the network capacity $n\lambda$ is

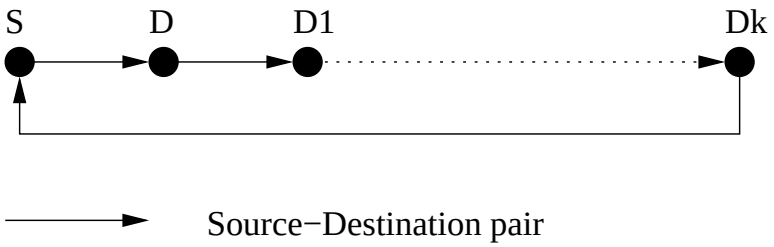

Figure 3.10: Example of a cycle formed by source-destination pairs under the permutation traffic model.

given by,

$$n\lambda = O\left(\frac{Wn}{c\max\limits_{i}(n_i)}\sqrt{\frac{n}{\log n}}\right) \tag{3.10}$$

We now estimate the value of $\max\limits_{i}(n_i)$. Recall our requirement that if a source node S is assigned to a channel, then its corresponding destination D should be assigned to the same channel. In turn, the destination of node D, say D1, should also be assigned the same channel. This process continues till the destination of one of the nodes is the first node S. Therefore, the source-destination assignments form a cycle as shown in Figure 3.10. Hence, the value of $\max\limits_{i}(n_i)$ is equal to the size of the largest cycle in a random permutation.

It is well-known [43] that the probability a random permutation has a cycle of length $m, m > \frac{n}{2}$, is $\frac{1}{m}$. Therefore, the probability that a permutation has a cycle of length greater than $\frac{n}{2}$ is given by [44],

$$\begin{aligned}
\text{Prob(cycle length greater than } \tfrac{n}{2}) &= \sum_{m=\frac{n}{2}+1}^{n} \frac{1}{m} \\
&\simeq \ln(n) - \ln\left(\frac{n}{2}\right) \quad \text{(for large n)} \\
&= \ln(2) \\
&\simeq 0.69
\end{aligned}$$

Therefore, $\max\limits_{i}(n_i)$ is at least $\frac{n}{2}$ with a non-zero probability exceeding a constant (independent of n). This implies that an upper bound on network capacity can be obtained by replacing $\max\limits_{i}(n_i)$ by $\frac{n}{2}$ in Equation 3.10 giving,

$$n\lambda = O\left(\frac{W}{c}\sqrt{\frac{n}{\log n}}\right) \tag{3.11}$$

This bound can be shown to be tight. A simple construction is to assign all nodes to a common channel, independent of the total channels available. Then, the common channel can support a data rate of $\frac{W}{c}$, and the network can be operated using the Gupta and Kumar construction for a single channel network. This construction yields the same capacity as specified by the upper bound in Equation 3.11, proving the upper bound is tight.

Comparing Equation 3.11 with Theorem 11, we can see that keeping the single interface at a node fixed results in a capacity loss. The loss can be as large as $\frac{1}{c}$ in the first capacity region (when $c = O(\log n)$). Therefore, this clearly suggests that if each node has a single interface, switching interfaces is necessary to avoid capacity degradation.

The random traffic model used earlier in the paper requires each node to randomly select a destination. Consider a graph built from the random traffic model, where vertices are nodes in the network, and two vertices are connected by an undirected edge if their corresponding nodes form a source-destination pair. In this graph, each vertex has an average degree of 2 (the graph has n edges, resulting in an average vertex degree of 2). A similar graph, called the random graph [45], can be constructed by taking n vertices and choosing every edge between vertices with a probability of $\frac{2}{n}$. The resultant graph also has an average vertex degree of 2, but is not identical to the graphs formed by the random traffic model. It has been shown that random graphs have a connected component of size $\Theta(n)$ when their average degree is greater than 1. Therefore, if source-destination pairs where chosen using the random graph approach, even then the network capacity would follow Equation 3.11. We speculate that the bound of Equation 3.11 also applies to the random traffic model considered earlier in the paper, though we do not have a proof to support the conjecture.

3.6.2 Lower bound with two fixed interfaces: permutation routing model

In the previous section, we showed that if nodes have a single interface, and if channel assigned to the interface is fixed, then there is a capacity degradation. In this section, we show that if all

nodes have two interfaces, and even if channels assigned to the interfaces are fixed, there is no loss in capacity, under the permutation traffic model. We designate the first interface at each node as the "primary interface", and the channel assigned to the first interface as the "primary channel". Similarly, the second interface is designated as the "secondary interface". The key idea here is that different nodes in a cell are assigned the primary channels such that in a cell, all channels have the same number of primary interfaces fixed to them. The secondary interface of a node is fixed to the primary channel of the node's destination. Data is sent from a source to a destination on the primary channel of the destination (over all hops). In the rest of this section, we show that this construction is feasible and achieves the same capacity as a network with one interface that can switch.

As before, the surface of the unit torus is divided into square cells each of area $a(n)$. We assume that c is at most n. Since the destination bottleneck constraint is not present in the random permutation model, we choose the area $a(n)$ slightly differently from before. Specifically, we set $a(n) = \max(\frac{100 \log n}{n}, \frac{c}{n})$. From Lemma 4, we know that each cell has $\Theta(na(n))$ nodes with high probability. Therefore, every cell has at least $k_2 na(n)$ nodes, where k_2 is a constant. If $k_2 na(n) < c$, we increase the area $a(n)$ by a factor of $\frac{c}{k_2 na(n)}$, and this factor can always be bounded by a constant (because $c = \Theta(na(n))$). The scaling ensures that every cell has at least c nodes. From now on, we assume that we are considering the scaled cells. Within each (scaled) cell there are at most $k_3 na(n)$ nodes, where k_3 is a constant. We number these nodes from 1 to $k_3 na(n)$, and assign node i to channel $(i \bmod c) + 1$. With this assignment there are $n_c = \Theta\left(\frac{na(n)}{c}\right)$ nodes per channel in each cell.

We use a simpler routing strategy (based on [31]) compared to what was used earlier in this chapter. Figure 3.11 shows the routing scheme. A random cell is chosen as the origin of a cell co-ordinate system. Each cell is assigned X and Y co-ordinates, such that the co-ordinate values change by 1 per cell (along each axis from the origin). To route from a node in a cell with co-ordinates (x_1, y_1) to a node in a cell with co-ordinates (x_2, y_2), the packets are first sent along the row containing the source till it intersects with the column containing the destination (i.e., follow along the row till the X-coordinate of the cell is x_2). After that, packets are sent along the column containing the destination till the destination cell is reached.

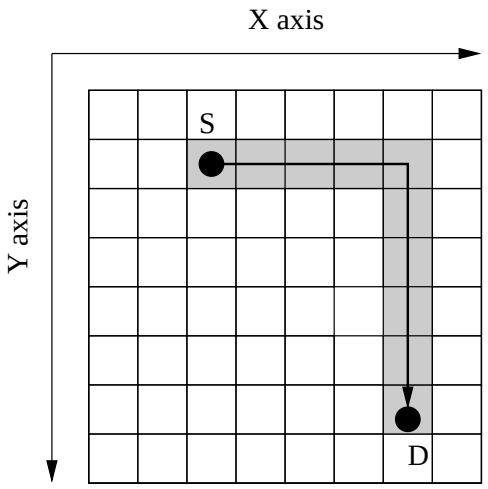

Figure 3.11: Routing through cells: Packets are routed first along a row till the destination column is reached, and then along the column to the destination cell.

Let us consider the traffic going through some cell L on some channel i. There are $\frac{1}{\sqrt{a(n)}}$ cells per column, and therefore, each column will have $\frac{n_c}{\sqrt{a(n)}}$ nodes on channel i. Since each node is the destination of exactly one flow under the permutation routing model, cell L will forward at most $\frac{n_c}{\sqrt{a(n)}}$ flows that are headed along the column containing L toward their destinations.

The network has $\frac{n_c}{a(n)}$ total destination nodes (there are $\frac{1}{a(n)}$ cells, and at most n_c nodes on a channel in each cell) receiving data on channel i. We now want to bound the number of source nodes sending data on channel i on any row. Since node locations are chosen independently at random, it is equally probable that the source node corresponding to a destination node is in any given row. Therefore, the number of source nodes in any given row may be modeled as a "Balls and Bins" problem where $\frac{n_c}{a(n)}$ balls are thrown into $\frac{1}{\sqrt{a(n)}}$ bins. Results from [37] show that if x balls are thrown into y bins, and $x \geq y \log y$, then each bin has $\Theta\left(\frac{x}{y}\right)$ balls w.h.p. Since in our case $x = \frac{n_c}{a(n)}, y = \frac{1}{\sqrt{a(n)}}$, and $n_c \geq 1$, implying $x \geq y^2$, each row will have $\Theta\left(\frac{n_c}{\sqrt{a(n)}}\right)$ source nodes. Therefore, as each source node originates one flow, the number of flows on any channel that pass through a cell L while traversing the row containing L (i.e., flows originate at some source node along the row containing L), is bounded by $\Theta\left(\frac{n_c}{\sqrt{a(n)}}\right)$.

Adding the flows along rows and columns, the total flows on any channel i passing through any cell L is $\Theta\left(\frac{n_c}{\sqrt{a(n)}}\right)$. Since each cell has n_c nodes on channel i, if the flows are carefully

balanced across nodes, the number of flows per node is given by $\Theta\left(\frac{1}{\sqrt{a(n)}}\right)$. Using the Gupta and Kumar [14] construction for a single channel, each node receives a fraction $\Theta\left(\frac{1}{n_c}\right)$ of the channel time (as there are $\Theta(n_c)$ nodes on any channel in a neighborhood). Therefore, each flow receives a throughput of $\Theta\left(\frac{W}{c}\frac{\sqrt{a(n)}}{n_c}\right)$ bits/sec. Substituting for n_c, and multiplying by n, the network capacity is $\Theta\left(W\frac{1}{\sqrt{a(n)}}\right)$ bits/sec. Substituting for $a(n)$, we have the following theorem:

Theorem 12. *The capacity of a random network with two fixed interfaces per node under channel model* 1*, and permutation traffic model, is as follows:*

1. *When c is $O(\log n)$, $a(n) = \Theta\left(\frac{\log n}{n}\right)$, and the network capacity is $\Omega\left(W\sqrt{\frac{n}{\log n}}\right)$ bits/sec.*
2. *When c is $\Omega(\log n)$ and also $O(n)$, $a(n) = \Theta\left(\frac{c}{n}\right)$, and the network capacity is $\Omega\left(W\sqrt{\frac{n}{c}}\right)$ bits/sec.*

The construction presented above could be easily generalized when more than two interfaces are available (by grouping interfaces using Lemma 2). Similarly, the constructions can be extended to the scenario with $c = \Omega(n)$ channels (by applying the earlier construction, but using only n channels), for which the network capacity is given by $\Omega\left(\frac{Wn}{c}\right)$ bits/sec. Comparing with Theorem 11, we see that there is no loss in capacity even if interfaces are fixed, provided the number of interfaces is doubled.

3.6.3 Lower bound with two fixed interfaces: random routing model

It can be shown that there is no loss in network capacity even under the random traffic model by using the construction techniques from [42]. In this approach, as before, the primary interfaces in each cell are carefully assigned to channels to balance the interfaces across all channels. The secondary interface at each node is independently assigned to a random channel, which is the key difference from the earlier construction. Traffic is initially sent from the source to the destination on the secondary channel, say s (at each hop, a node with primary interface tuned to channel s relays the packet). When the packet is within $c \log n$ hops of the destination[8], then the packet enters a transition phase. During the transition phase, if the packet is at a node that has its primary channel

[8]If a route requires fewer than $c \log n$ hops, then the route length is intentionally increased by using a detour [42].

on s and the secondary channel on the destination's primary channel, say d, then the packet is sent out on channel d. After the packet transitions to the destination's primary channel, it is relayed on that channel till it reaches the destination. This construction can be shown to achieve the same network capacity as a network where interfaces can switch (see constructions in [42] for more details). Hence, even under the random traffic model, two fixed interfaces per node is sufficient to achieve asymptotically optimal capacity.

3.7 Discussions

The theoretical analysis has yielded the capacity of wireless networks with the number of channels varying across a wide range. The region where the number of channels is scaled as $O(\log n)$ seems to be of immediate practical interest, since the number of channels provisioned for in current wireless technologies is not too large. However, there are many recent efforts aimed at utilizing frequency spectrum in higher frequency bands, where significantly larger bandwidth is available for use. For example, there is around 7 GHz of spectrum available for unlicensed use in the 60 GHz band [6], whereas the total bandwidth used in current wireless technologies, such as IEEE 802.11, is less than 500 MHz. The bandwidth that may become available in higher frequency bands can be split up into a large number of channels, and therefore the region with number of channels greater than $\Omega(\log n)$ may be of practical interest in the near future.

The capacity analysis has shown that a single interface may suffice for random networks with up to $O(\log n)$ channels. The capacity-optimal lower bound construction used to support the above claim is based on certain assumptions, all of which may not be satisfied in practice. For example, we assume that interface switching delay is zero, transmission range of interfaces can be carefully controlled, and there is a centralized mechanism for co-ordinating route assignment and scheduling. In addition, the theoretical analysis derives asymptotic results, and capacity can be improved by constant factors in the lower bound constructions by using multiple interfaces. From Section 3.5, we note that when interface switching delay is not zero, having more than one interface may be beneficial. Furthermore, protocol design has identified many benefits of using at least two interfaces at each node, such as allowing full-duplex transfer, and simplifying the development of distributed

protocols for utilizing multiple channels, as seen in later chapters.

Our simulation and testbed experiments (reported in later chapters) have shown that having more than one interface may be beneficial in practice. However, these experiments do not prove multiple interfaces are necessary for obtaining all the observed performance improvement. In addition, our simulation results also show that it is not necessary to have one interface per channel to utilize all the channels, and in fact even many (e.g., 12) channels can be fully utilized by using only two interfaces, which validates the theoretical claim. Therefore, in practice, the theoretical claim that a single interface suffices with $O(\log n)$ channels is reasonably accurate, with the caveat that additional interfaces may be useful in simplifying protocol design and hiding switching delay.

In summary, in this chapter we have derived the lower and upper bounds on the capacity of static multichannel wireless networks. We have considered wireless networks having c channels, and $m \leq c$ interfaces per node. Each interface is capable of selecting appropriate transmission power, and lower bound constructions require global knowledge. Under this model, we have shown that in an arbitrary network, there is a loss in the network capacity when the number of interfaces per node is smaller than the number of channels. However, we have shown that in a random network, a single interface may suffice for utilizing multiple channels, as long as the number of channels is scaled as $O(\log n)$. We have then considered the impact of non-zero interface switching delay on capacity, and shown that in a random network with up to $O(\log n)$ channels, interface switching delay has no impact on capacity, provided each node is provisioned with a few extra interfaces. We have also considered the scenario where after an initial channel assignment, interfaces are not allowed to switch channels. Under this model, we show that if each node has only one interface, then there is a loss in network capacity. However, the capacity loss can be eliminated by providing an additional interface at each node, which shows that it may be possible to develop protocols which do not require interface switching, albeit at the cost of using extra hardware.

CHAPTER 4

PROTOCOL DESIGN OVERVIEW

Capacity analysis has identified the benefits of using a large number of wireless channels even if only a few interfaces are available. Constructions used to prove capacity bounds also provide centralized interface assignment and routing algorithms that are order optimal. However, order optimal algorithms may not offer the best performance in small networks. Furthermore, some of the assumptions made in the capacity analysis, such as the ability to set transmission power to any desired value, are not always feasible in practice. In addition, practical implementations require distributed protocols that will have to work with incomplete topological and traffic information. In this chapter, we transition towards the development of *practical* multichannel protocols that are based on the insights from capacity analysis.

In the rest of this chapter, we first formulate the requirements under which the protocols in this thesis are designed. We then survey related research, and show how existing research does not meet the desired protocol requirements. After that we present insights from capacity analysis that are useful for protocol design, explain the high-level design choices that we make, and conclude with a description of the system architecture.

4.1 Protocol requirements and assumptions

The protocols proposed in this thesis are designed for networks where multiple channels are available. The key goal of the protocols is to effectively utilize the available channels. The network is assumed to operate in a multihop fashion by routing packets over multiple wireless hops. The nodes in the network could be static or mobile. There are several kinds of multihop networks depending on the application for which the network is designed. Examples include ad hoc networks, mesh

networks, and sensor networks. Our goal in this thesis is to develop general protocols that are not restricted to any specific kind of multihop network. With this background, we define the set of requirements that our protocols are designed to meet:

1. Flexibility to operate with different number of channels and interfaces is desired. As we discussed earlier, the number of available channels may change in the future. Similarly, future hardware may combine several interfaces into an integrated hardware device. Consequently, flexibility to operate with different number of channels and interfaces is an important design goal.

2. Flexibility to operate with different kinds of traffic patterns is desired. For example, traffic in mesh networks may be mostly directed to few gateway nodes, while ad hoc networks may require communication between arbitrary nodes. In this thesis, we assume that the traffic pattern is unknown, and hence the protocols should be general enough to allow any pair of nodes in the network to communicate with each other.

3. Maintain the network connectivity available in a single channel network. For example, a multichannel solution could reduce network connectivity if nodes that are neighbors in a single channel network are permanently assigned to different channels, thereby preventing communication between them. In the worst case, reduction in network connectivity could partition the network. The need to maintain the level of connectivity available in a single channel network is motivated by the lack of knowledge of traffic requirements. Reduced connectivity could prove to be significantly worse for certain traffic patterns. Therefore, in the absence of knowledge of traffic patterns, it is prudent to maintain the connectivity available in a single channel network.

4. Operation with off-the-shelf hardware is desired. Although, future hardware may be more flexible, we base our protocols on the features currently available. This is a pragmatic requirement to ensure that the protocols can be implemented in a testbed. This goal precludes solutions that require changes to hardware. Since most of the MAC features are currently built into interface hardware, it is not desirable for the protocols to require changes to MAC. Driven by this requirement, protocols in this thesis are built above the MAC. In addition,

the inability to modify hardware implies it is difficult to achieve tight time synchronization between different interfaces (and nodes). Motivated by this constraint, we carefully design the protocols to not require tight time synchronization.

In addition to these requirements, the primary goal of the protocols in this thesis is to improve network capacity. We do not explicitly consider other performance metrics, such as fairness.

4.2 Related work

Several researchers have proposed MAC protocols based on IEEE 802.11 for utilizing multiple channels. Nasipuri et al. [46, 47], and Jain et al. [48] have proposed a class of protocols where all nodes have an interface on each channel. The protocols differ in the metric used to choose a channel for communication between a pair of nodes. The metric may simply be to use an idle channel [46], or the signal power observed at the sender [47], or the signal power observed at the receiver [48]. Adya et al. [49] have proposed a link-layer solution for striping data over multiple interfaces, but their solution is designed for the scenario with the number of interfaces per host equal to the number of channels. These protocols are expensive to implement as they require each node to have one interface per channel.

Wu et al. [50] have proposed a MAC layer solution that requires two interfaces. One interface is assigned to a common channel for control purposes, and the second interface is switched between the remaining channels and used for data exchange. RTS/CTS packets (as in IEEE 802.11) are exchanged on the control channel, and the exchange also determines the appropriate data channel to use for subsequent DATA/ACK exchange. A similar control channel approach has been studied in Ravichandran's thesis [51]. Hung et al. [52] have also proposed a two-interface solution that uses a channel load-aware algorithm to choose the appropriate data channel to use for DATA/ACK exchange. Chen et al. [53] have proposed another two-interface solution that also uses a dedicated control channel for RTS/CTS exchange. Ramachandran et al. [54] have proposed a multichannel protocol which uses a common channel for maintaining network connectivity. Gong et al. [55, 56] have proposed a joint channel assignment and routing protocol for multichannel wireless networks. The underlying multichannel MAC protocol uses a control channel for RTS/CTS exchange, before

data is transferred on any of the remaining channels. While all these proposals require only two interfaces to support any number of channels, the common control channel may become a bottleneck to performance. Since RTS/CTS exchange precedes *each data transmission*, this approach does not scale when the number of data channels is large.

So et al. [57] have proposed a multichannel MAC protocol, called MMAC, which requires a single interface at each node. Nodes in the network rendezvous on a common channel periodically. During the rendezvous interval, the channel to be used till the next rendezvous interval is selected, based on traffic and channel load. MMAC requires tight synchronization and can significantly increase the delay over multihop routes. Other protocols for exploiting multiple channels with a single interface have been explored in So's thesis [58].

Bahl et al. [59] have proposed SSCH, a link layer solution that uses a single interface. SSCH can be extended to utilize multiple interfaces as well. SSCH requires individual nodes to hop among channels, based on a previously advertised schedule. A node stays on a channel for a specified slot time (10 ms), and the delay incurred in switching an interface from one channel to another is expected to be small (80 μs). A key requirement for achieving good performance with SSCH is to use short slot times, which in turn requires fast interface switching. However, currently available commercial hardware requires of the order of a millisecond to switch channels, which is an order of magnitude higher than what SSCH assumes. A key design goal of this thesis is to realize good performance even when interface switching delay is fairly large, thereby allowing protocols to be implemented with currently available hardware. In addition, SSCH does not consider routing issues with multichannel networks.

In the context of wired networks, Marsan et al. [60] have studied the performance of multichannel CSMA/CD MAC protocols, and shown that significant reduction in delay average and variance is possible even when the number of interfaces is less than the number of channels. This thesis addresses a similar question with multichannel CSMA/CA based wireless networks.

Tang et al. [61] have considered the joint channel assignment and QoS routing problem for multichannel networks. Polynomial time centralized algorithms for this problem have been presented in the paper. Since the solution is not distributed, the algorithms are difficult to use in practice.

In addition, the paper is more focused on quality of service issues, while this thesis focuses on providing best-effort service.

Existing routing protocols for multihop networks, such as DSR [62] and AODV [63], can support multiple channels at each node, provided each channel has a dedicated interface. In addition to the need for a large number of interfaces, those protocols typically select shortest-path routes, which are often not suitable for multichannel networks [4].

Shacham et al. [64] have proposed an architecture for multichannel networks that uses a single interface. Each node has a default channel for receiving data. A node with a packet to transmit has to switch to the channel of the receiver before transmitting data. However, the proposal does not consider the impact of switching delay. Furthermore, the architecture does not guarantee network connectivity and requires complex coordination. So et al. [65] have proposed a routing protocol for multichannel networks that also uses a single interface at each node. Their routing protocol requires tight synchronization between nodes, which is difficult to achieve in practice.

There are a few routing proposals specifically designed for multichannel wireless networks. Raniwala et al. [35,66] have proposed routing and interface assignment algorithms for multichannel multi-interface networks. Similar to this thesis, they also consider the scenario wherein the number of available interfaces is smaller than the number of available channels. However, their solution is designed specifically for use in those mesh networks where all traffic is directed toward specific gateway nodes. In contrast, the protocols in this thesis are designed for more general ad hoc networks (as well as mesh networks with significant intra-mesh traffic), where potentially any node may communicate with any other node.

Draves et al. [4] have proposed a new routing protocol, called multi-radio link quality source routing (MR-LQSR), for multichannel ad hoc networks that ensures "channel diverse" routes are selected. MR-LQSR has been designed with the assumption that the number of interfaces per node is *equal* to the number of channels. In contrast, this thesis focuses on the more general scenario where the number of available interfaces may be smaller than the number of available channels, and interface switching is required to utilize all the channels. We present a new routing metric in this thesis that accounts for interface switching, and builds upon the metric used by MR-LQSR.

4.3 Insights from capacity analysis

In this section, we discuss insights for protocol design derived from the capacity analysis. The insights are useful in identifying good architectural principles for building multichannel networks, and sound approaches for interface assignment and routing in multichannel networks.

4.3.1 Network architecture

One architecture used in the past [4], when there are c channels, and $m < c$ interfaces per node, is to use only m channels in the network. Every node permanently fixes its m interfaces to a common set of m channels. The benefit of this architecture is that explicit interface assignment is not required. Although this approach does not require new multichannel protocols (other than an initial assignment), many channels are left unused, especially when the number of interfaces per node is significantly smaller than the number of available channels. Capacity analysis clearly shows this approach to be suboptimal, and instead recommends using all the available channels even if it requires interfaces to be switched across channels.

When the interfaces may have to be switched across channels, an interface assignment protocol is needed to schedule (over time) the channels assigned to an interface. In addition to interface assignment, route selection has to be carefully done to ensure that channels are fully utilized. One possibility is to develop a joint interface assignment and routing protocol for achieving optimal performance. Such an approach requires cross-layer interaction, and may lead to implementation difficulties in practice. However, the constructions used in the lower bound proof of capacity analysis separate the process of interface assignment and routing. Since the lower bound constructions are order optimal, this suggests that even if interface assignment and routing are implemented separately, it may be possible to achieve good performance. This insight suggests that the traditional approach of separating out routing and interface assignment continues to be a good strategy in multichannel networks.

4.3.2 Interface assignment and power control

Earlier works on capacity [14] suggest that the common transmission power for the network should be set to the smallest power that keeps the network connected. Capacity analysis showed that in a multichannel network, the transmission power should be chosen to keep the network connected, and in addition ensure that the average number of nodes in every interference neighborhood is equal to the number of channels. This implies that as the number of available channels increase, larger transmission power (than that is needed for connectivity) should be used. In practice, it may not be possible to choose arbitrary power levels, as most interface hardware today only allow a small set of power levels. Therefore, in practice, this insight suggests that at the network design phase, a common transmission power can be chosen from the available power levels based on the network density and the number of channels expected to be available. Later, if more channels become available, a new power level may be a necessary.

After the transmission power has been fixed, capacity analysis suggests that channels should be assigned such that the number of nodes assigned to any channel in a given neighborhood is (almost) the same. Intuitively, the goal of the interface assignment is to distribute traffic evenly across all channels. In large networks with random traffic patterns, all nodes end up handling similar amounts of data. Therefore, balancing nodes across channels corresponds to balancing traffic across channels. In small networks, balancing nodes across channels may not balance traffic as well. However, it may not be possible a priori to have knowledge of traffic patterns. Therefore, the balancing strategy suggested by the capacity analysis may be a good strategy in practice as well.

4.3.3 Routing

Traditional routing protocols for single channel multihop networks (c.f., [62,63,67]) have considered the use of metrics that reduce the hop count, or other notions of cost of the route. However, these metrics do not take into account other routes that may be already in use in the network. The lower bound construction in the capacity analysis requires the distribution of routes among nodes in a given neighborhood to ensure full utilization of multiple channels. In a single channel network, load balancing within a neighborhood is sometimes used to balance energy consumption across nodes, or

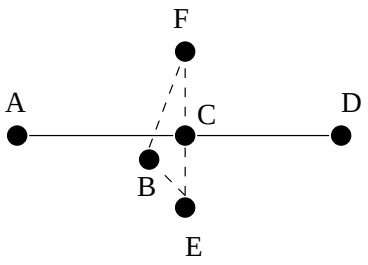

Figure 4.1: Impact of route selection on effective utilization of multiple channels

to improve resilience of the network. However, route distribution in the same neighborhood is not required in single channel networks for maximizing capacity. Therefore, capacity analysis suggests that routing protocols may have to be redesigned for multichannel networks.

Figure 4.1 illustrates a scenario that highlights the need for specialized routing protocols for multichannel networks. In the figure, node A is communicating with node D using route A-C-D. Node E wishes to communicate with node F, and either of B or C can be used as the intermediate node. Assume that all nodes have a single interface, and assume that C and B can relay at most w bytes per second. If node C is chosen as the intermediate node, then node C has to forward data along both routes A-C-D and E-C-F, and the throughput received by each flow is at most $w/2$. On the other hand if node B is chosen as the intermediate node, then both routes A-C-D and E-B-F can be simultaneously used (provided different channels are used on each hop of routes A-C-D and E-B-F), and each flow receives a rate of w. Although this example assumed that each node had a single interface, similar issues arise even when multiple interfaces are available.

The above scenario highlights the need for the routing protocol to appropriately *distribute* routes among nodes in the neighborhood. In the case of single channel networks, the throughput obtained is the same whether B or C is chosen as the intermediate node. When a single channel is available, and say, when C is transmitting a packet along route A-C-D, B cannot transmit a packet even if it is chosen as the intermediate node (as the common channel is busy). Consequently, routing protocols designed for single channel networks do not need to distribute routes within a neighborhood. However, to exploit the benefit of multiple channels, it may be important for a routing protocol to ensure that routes are carefully distributed in the network.

4.4 Design choice: Using multiple interfaces

We propose to design protocols for multichannel networks under the assumption that multiple interfaces are available at each node. In this section, we justify our design choice of using multiple interfaces. In most multihop networks, a single channel is used, and therefore a single interface suffices. However, when multiple channels are available, having more than one interface is beneficial, as we describe next. Reducing hardware costs has made it feasible to equip each node with multiple interfaces. For example, nodes in our testbed are equipped with at least two interfaces.

Capacity analysis has shown that when switching delay is negligible, or when there are no latency constraints, a single interface per node may suffice in achieving optimal capacity in random networks. However, in practice, switching delay is often non-negligible, and applications typically expect end-to-end latency to be reasonably small. Under these constraints, our analysis shows that multiple interfaces per node may be required for achieving optimal capacity. Apart from the theoretical need for multiple interfaces to achieve asymptotically optimal capacity, many practical concerns, described below, motivate the use of multiple interfaces as well.

When using a single interface per node, if the interfaces of two nodes are on different channels, then they cannot communicate. For reducing synchronization requirements and overheads, each interface has to stay on a channel for many packet transmission durations (100 ms in [57] and 10 ms in [59]). As a result, when packets are traversing multihop paths, packets may be delayed at each hop, unless the next hop is on the same channel as well. Therefore, when a single interface is used, there is an increase in the end-to-end latency if different hops along a route are on different channels. Otherwise, if most hops are on the same channel, transmissions on consecutive hops interfere, reducing the maximum capacity. In either case, TCP throughput is significantly affected, since it depends on the end-to-end delay and the bottleneck link throughput.

When at least two interfaces are available, we propose to fix one interface on a channel to greatly simplify coordination, while switching the second interface (based on traffic requirements) to avoid delaying a packet at each hop. We defer discussion of the details of the protocol to a later chapter, but multiple interfaces allow us to derive both simplicity in coordination and minimal delays.

Our approach of keeping one interface at each node fixed on a specified channel and switching the second interface turns out to be optimal in the asymptotic setting. For example, in the transmission scheduling scheme used in the lower bound construction of capacity analysis, it suffices for a node to always transmit on a specific channel without having to switch channels for different packets. However, a node may have to switch channels for receiving data. An alternate construction is to use a scheduling scheme which ensures that a node receives all data on a specific channel, but may have to switch channels when sending data. It can be easily shown that the alternate construction is equivalent to the construction we have actually used. Therefore, our choice of keeping one interface fixed and switching the second interface is asymptotically optimal.

Another benefit of using two interfaces is the ability to receive and transmit data in parallel. Half-duplex wireless interfaces cannot simultaneously transmit and receive data. However, when multiple (say two) interfaces and multiple channels are available, while one interface is receiving data on one channel, the second interface can simultaneously transmit data on a different channel. In many cases, this can double the maximum throughput achievable on a multihop route. Our protocols exploit this benefit of using multiple interfaces as well.

4.5 Proposed architecture

In this thesis, we separate interface assignment from routing. As we discussed earlier in Section 4.3, such a separation does not degrade performance (in the asymptotic setting). In addition to managing interface assignment, protocol support is needed for buffering packets and managing interface switching. We designate the combined protocol that implements interface assignment, buffering and interface switching, as the "interface management protocol". Under this setting, we have an interface management protocol to manage the use of multiple interfaces, and a routing protocol that interacts with the interface management protocol to select good routes.

Separating interface management from routing significantly simplifies protocol design. Interface switching can occur on the time scales of a few packet transmissions, while routing changes happen less frequently. Therefore, separating the two components offers a cleaner design. Interface management protocol operates primarily at the link layer, while the routing protocol operates primarily

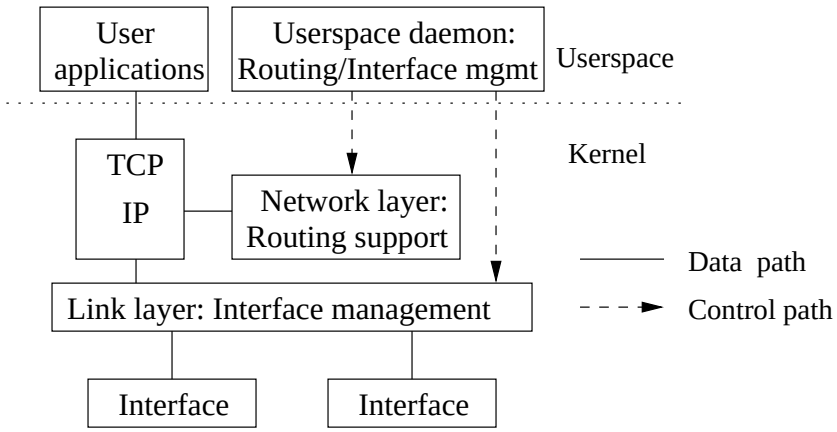

Figure 4.2: Proposed network architecture

at the network layer. Therefore, separating the two protocols also reduces cross layer interactions. The interface management protocol completely hides the complexity of managing multiple channels and interfaces from the higher layers. Therefore, a further benefit of the separation is the possibility of using different routing protocols (including existing protocols) over the interface management protocol.

Figure 4.2 shows the protocol architecture. Based on the principle of minimizing kernel code, our protocol implementation builds only the required interface management support in the link layer of the kernel. Some of the time-critical components of routing (used for forwarding data) are built into the network layer of the kernel. The less time-critical components of the interface management and routing protocols are implemented as a userspace daemon. We have implemented this architecture in a Linux-based testbed, and more details of our implementation are in Chapter 7. In Chapter 5 we will describe the interface management protocol. Chapter 6 presents a new multichannel routing metric (MCR) designed for multichannel operation, and an on-demand routing protocol that uses the new metric.

CHAPTER 5

INTERFACE MANAGEMENT PROTOCOL

In this thesis, we have separated interface management from routing. As discussed earlier, when the number of available interfaces is smaller than the number of available channels, an interface assignment strategy is required to assign interfaces to specific channels. In addition, when a packet is handed down to the interface management protocol, the channel on which the packet has to be sent out may not have an interface assigned to it at that time. Such a packet will have to be buffered till an interface becomes available on the desired channel. Therefore, the interface management protocol has to buffer packets when necessary, and schedule interfaces to switch to channels having pending packets. Switching interfaces frequently, with respect to the interface switching delay, can be expensive. Consequently, the frequency of interface switching has to be carefully chosen to achieve acceptable packet delays while minimizing overheads. Interface switching also has to be carefully done to ensure that the neighbors of a node can communicate with it on-demand, thereby retaining the network connectivity available in a single channel network.

We will first classify existing interface assignment strategies, and motivate why those strategies are not sufficient for multichannel networks with large number of channels and few interfaces. We then present a new interface assignment strategy, and incorporate the strategy into a new interface management protocol. We then present a thorough evaluation of the interface management protocol.

5.1 Need for a new interface assignment strategy

When the number of interfaces per node is smaller than the number of channels, we need an interface assignment protocol that assigns interfaces to channels, and over time decides how interfaces are

switched across channels. There are several interface assignment strategies that may be used [36]. In this section, we classify the interface assignment strategies, identify the shortcoming of previously proposed strategies, and propose a new interface assignment strategy that is superior to past works.

We classify interface assignment strategies, based on how frequently interfaces are switched, into static, dynamic, and hybrid strategies.

1. Static Assignment: Static assignment strategies assign each interface to a channel either permanently, or for "long intervals" of time, where "long interval" is defined relative to the interface switching and packet transmission time. For example, [4,66] use static interface assignment. Static assignment can be further classified into two types:

1. Common channel approach: In this approach, interfaces of all nodes are assigned to a common set of channels (cf. [4]). For example, if two interfaces are used at each node, then the two interfaces are assigned to the same two channels at every node. The benefit of this approach is that the connectivity of the network is the same as that of a single channel approach. However, the main problem with this approach is that when a dedicated interface is not available for each channel, many channels are left unused. Capacity analysis has shown that such a strategy is suboptimal.

2. Varying channel approach: In this approach, interfaces of different nodes may be assigned to a different set of channels (cf. [66]). With this approach, two nodes which are in the communication range of each other may not be able to communicate if their interfaces are tuned to different channels. Therefore, this approach may reduce connectivity of the network. Consequently, there is a possibility that the length of the routes between nodes may increase. In addition, unless the interface assignment is carefully done, network partitions may arise.

Static assignment strategies are useful when the interface switching delay is large, or there are nearly as many interfaces as channels. Therefore, if the number of available interfaces is equal to the number of available channels, interface assignment problem reduces to a static assignment. Static assignment strategies do not require explicit co-ordination among nodes (except perhaps to occasionally assign interfaces for long intervals of time) before data communication. With static assignment, nodes that share a channel on one of their interfaces can directly communicate with

each other (if within range), while others cannot. Thus, the effect of static channel assignment is to control the network topology by deciding which nodes can communicate with each other.

2. Dynamic Assignment: Dynamic assignment strategies allow any interface to be assigned to any channel, and interfaces can frequently switch from one channel to another. In this setting, two nodes that need to communicate with each other use a co-ordination mechanism to ensure that they are on a common channel at some point of time. The benefit of dynamic assignment is the ability to switch an interface to any channel, thereby offering the potential to cover many channels with few interfaces.

The key challenge with dynamic switching strategies is the need to co-ordinate the process of switching interfaces among the nodes in the network. When a node X switches its interface from one channel to another, neighbors of X should not continue to assume that X has an interface on the prior channel. If all interfaces dynamically switch, then to avoid the need for co-ordination among nodes on a per-packet basis, neighboring nodes should be aware of each other's switching schedule[1]. Such awareness has been implemented in past works by either using a periodic rendezvous on a common channel to exchange future switching schedules [57], or by using well-known pseudo-random sequences [59] to control channel switching. Both these approaches require tight time synchronization between nodes, which can be difficult to implement in practice. Therefore, although dynamic approaches offer the most flexibility, they may not always be practical to use in real networks.

3. Hybrid Assignment: Hybrid assignment strategies combine static and dynamic assignment strategies by applying a static assignment to some interfaces and a dynamic assignment to other interfaces. Hybrid strategies can be further classified based on whether the interfaces with static assignment use a common channel approach, or a varying channel approach. Hybrid assignment strategies are attractive because static assignment requires little co-ordination, while dynamic assignment offers significant flexibility.

A version of the hybrid interface assignment using the common channel approach, called the "common control channel" approach, is quite popular [50, 52, 53, 55, 56]. In this approach, one

[1]Switching schedule can be used to calculate the channel assigned to an interface at any point in time.

interface at every node is tuned to a common control channel. The remaining interfaces can be switched among any of the remaining channels. Before a data packet is sent out, the channel to use for sending the data packet is negotiated on the control channel (using RTS-CTS exchange). Since all nodes share a common channel, two nodes in communication range can always reach each other. This ensures that full connectivity is maintained, yet all the available channels can potentially be used. However, a key drawback of this approach is that the common control channel becomes a bottleneck to performance [68], especially when the number of available data channels is large. Since it is quite likely that a large number of channels may be available, the common control channel approach is not a suitable choice.

Motivated by the shortcomings of past works, we propose a new interface assignment strategy. The new strategy may be classified as a hybrid interface assignment strategy with varying channel approach. The next section describes the proposed interface assignment strategy.

5.2 Proposed interface assignment strategy

Suppose that there are $m > 1$ interfaces available at each node[2]. We propose to divide the available interfaces into two disjoint subsets.

- *Fixed Interfaces:* Some $1 \leq f < m$ of the m interfaces at each node are assigned for long intervals of time to some f channels, and we designate these interfaces as "fixed interfaces", and the corresponding channels as "fixed channels".
- *Switchable Interfaces:* The remaining $m - f$ interfaces are dynamically assigned to any of the remaining $m - f$ channels (over short time scales), based on data traffic. These interfaces are designated as "switchable interfaces", and the channel to which a switchable interface is assigned to is called a "switchable channel".

Different nodes may assign their f fixed interfaces to a different set of f channels. It is possible for each node to use a different value of f and m, and it is also possible to vary f with time. This interface assignment strategy may be classified as hybrid interface assignment using the varying approach for the fixed channels.

[2]We will discuss later extensions to support $m = 1$ interface per node

Every node has at least one fixed interface assigned to a fixed channel. The key idea used to simplify co-ordination is to require a node X, with a packet to send to a node Y, to send the packet on one of the fixed channels of Y. If every node is aware of the fixed channels used by its neighbors, then explicit co-ordination is not needed before sending out a data packet. If a node has at least two interfaces, we ensure that at least one of its interfaces is set up as a switchable interface. When a packet has to be sent to a neighbor, and the node and its neighbor do not share any fixed channels, then the switchable interface is switched to a fixed channel of the neighbor before transmitting the packet. Therefore, when every node has at least two interfaces (which implies one of the interfaces is a switchable interface), a node can exchange data with any of its neighbors, irrespective of the fixed channels being used by the neighbors. This ensures that the connectivity with the proposed interface assignment strategy is the same as the connectivity in a single channel network. Furthermore, if different nodes use different fixed channels (the protocol for doing this is explained later), traffic load is distributed across channels, avoiding the bottlenecks that arise with the common control channel approach.

Figure 5.1 illustrates the operation of the proposed strategy. Assume that node A has packets to send to node C via node B. Assume that nodes A, B, and C have their fixed interfaces on channels 1, 2, and 3 respectively. Assume that initially node A has its switchable interface on channel 3, node B has its switchable interface on channel 1, and node C has its switchable interface on channel 2. In the first step, node A switches its switchable interface from channel 3 to channel 2, before transmitting a packet to node B, because channel 2 is the fixed channel of node B. Node B can receive the packet since its fixed interface is always listening to channel 2. In the next step, node B switches its switchable interface to channel 3 and forwards the packet to node C, which is received by node C using its fixed interface. Once the switchable interfaces are correctly set up during a flow initiation, there is no need to switch the interfaces for subsequent packets of the flow (unless a switchable interface has to switch to another channel for sending packets of a different flow).

In the next section, we describe the new interface management protocol [69] for implementing this strategy.

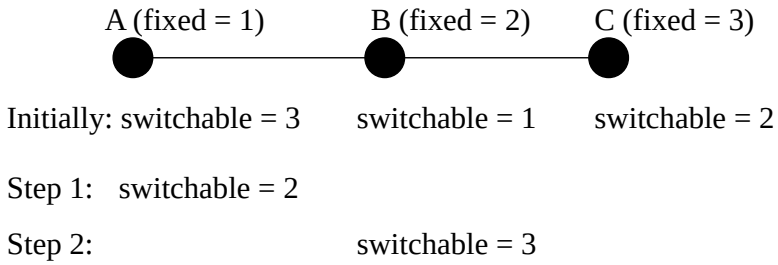

Figure 5.1: Example of interface management protocol operation with 3 channels, 2 interfaces

5.3 Interface management protocol

We assume that the available m interfaces at each node are partitioned into f fixed interfaces, and $m - f$ switchable interfaces. In the next section, we discuss strategies for partitioning the available interfaces into fixed and switchable interfaces. Fixed interfaces of a node are assigned to some fixed channels for long intervals of time. Each node maintains a *NeighborTable* containing the fixed channels being used by its neighbors. In Section 5.3.1, we describe the protocol used for exchanging data packets after the *NeighborTable* has been populated. The protocol for choosing fixed channels, and building the *NeighborTable* is described in Section 5.3.2.

5.3.1 Protocol for communication between nodes

The protocol for exchanging data packets between neighbors is implemented at the link layer, below the network layer and above the interface device drivers. This component has to look at every data packet that has to be sent out. Therefore, for efficiency reasons, this part of the interface management protocol has to be built as part of the kernel. We next describe the various components of the link layer protocol.

Inserting packets into channel queues:

Each channel is associated with a packet queue, as shown in Figure 5.2. If an unicast packet is received at the link layer for transmission, the fixed channel of the destination of the packet is looked up in the *NeighborTable*, and the packet is added to the corresponding channel queue. When a destination has multiple fixed channels, we select one of the fixed channels as the *preferred* fixed channel, and use that channel for sending data packets to the destination. The preferred fixed

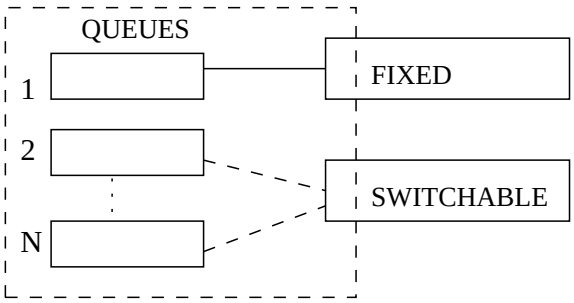

Figure 5.2: Illustration of queues associated with N channels and 2 interfaces

channel is changed periodically (this is done at the frequency of hello packet exchange, described later) to distribute load over the available fixed channels. An alternate possibility would be to stripe data to the destination over all the available fixed channels, but striping is non-trivial to implement since a naive approach may lead to packet reorderings. We have not explored the packet striping approach in our work, and defer it to a future study.

In single channel networks, a packet broadcast on the channel can potentially be received by all neighbors of the transmitter. However, when multiple channels are being used, a packet broadcast on a channel is received only by those nodes listening to that channel. Many higher-layer protocols (e.g., routing protocols) require broadcast packets to be received by all nodes in the neighborhood. Such neighborhood broadcast is supported in our protocol by transmitting the a copy of the broadcast packet on every channel. A copy of the broadcast packet is added to each channel's queue, and sent out when that channel is scheduled for transmission by the protocol.

Although sending a copy on each channel increases the cost of broadcast with increasing number of channels, the *cost per channel* remains same. For example, in a c channel network, each broadcast involves sending c copies of the packet on c channels, resulting in only 1 packet per channel. Thus, the overhead per channel is the same as in a single channel network. However, too many broadcasts may still be detrimental to performance, since switchable interfaces have to frequently switch channels to transmit each copy of a broadcast packet, degrading performance. Broadcast overheads can be reduced by sending broadcasts over a separate broadcast channel, equipped with a dedicated interface. Such an approach requires at least three interfaces at each node (one fixed, one switchable, one broadcast). An alternate approach for reducing overheads is to use *partial*

broadcasts [59], wherein broadcast packets are sent out only on a subset of available channels. Although the partial broadcast approach can reduce overheads, it may also reduce the number of nodes receiving a broadcast message, possibly reducing connectivity.

Servicing channel queues:

Fixed interfaces transmit packets queued up for transmission on their respective fixed channels. Packets are transmitted on all other channels using switchable interfaces. When a switchable interface is switched to a new channel, it is always switched to the channel that has the oldest queued data. This policy improves fairness, and reduces the worst case queuing delay. A switchable interface changes channels only when there are packets queued for another channel, and one of the following two conditions hold:

1. The switchable interface is on a channel with an empty queue.

2. The switchable interface has been on a channel for more than *MaxSwitchTime* duration. This condition prevents starvation of other queues.

Using a small value of *MaxSwitchTime* increases switching overhead (although the overhead increases only if data is present for multiple channels), while using too large a value increases end-to-end latency. In our simulations, we set *MaxSwitchTime* to be at least ten times the switching delay, which is sufficient to allow several packet transmissions per switch.

IEEE 802.11 maintains a virtual network allocation vector (NAV), set from overheard RTS-CTS transmissions, to track the channel busy intervals. When an interface is switched to a new channel, the virtual NAV on the new channel may not be correct because the node may have missed earlier RTS-CTS transmissions. Therefore, a node has to wait for a sufficient duration to ensure that it does not interfere with ongoing transmissions. For example, if RTS-CTS is enabled in the network, then the interface should defer for one maximum sized packet transmission. This strategy ensures that ongoing transmissions are protected from interference. In our simulations, after an interface switch, we wait for the time required to transmit one maximum sized packet.

5.3.2 Managing fixed interface

Fixed interface management involves two components - choosing the channels to be assigned to the fixed interfaces, and informing neighbors about the channels being used by the fixed interfaces. The interface management protocol has to ensure that fixed interfaces of nodes in a neighborhood are distributed across different channels. For example, suppose a node A uses channel 1 for its fixed interface. Then, all transmissions directed to A will be on channel 1. For balancing the usage of channels, it is beneficial if other nodes in the neighborhood use a different channel for their fixed interface.

Balancing fixed interfaces across channels does not necessarily balance traffic load, especially in scenarios where most traffic is directed to specific nodes. In such scenarios, explicitly balancing traffic might be beneficial [54]. However, it is often difficult to predict the traffic patterns a priori. Therefore, we have made a design choice of balancing fixed interfaces across channels. We believe that if most traffic is directed to certain nodes, then those nodes should be equipped with more fixed interfaces. With such an approach, balancing fixed interfaces can balance traffic as well.

Another reason for not balancing based on traffic load is because the load in the network could change frequently. For example, HTTP transfers are often less than a second long, and if such short-lived flows dominate the network traffic, then it may lead to frequent changes in channel load. Basing fixed channel selection policy on the network topology requires switching only when the topology changes, which can be of the order of tens to hundreds of seconds even with moderate mobility. Hence, we have opted to choose fixed channels based on the number of fixed interfaces using a channel. However, other policies can be easily incorporated into our protocol.

We propose a localized protocol for fixed interface management. Initially, nodes assign a random channel to each fixed interface, while ensuring that different fixed interfaces at a node use different channels. Recall that each node maintains a *NeighborTable* containing the fixed channels being used by its neighbors (initially, the table is empty). Nodes also maintain a *ChannelUsageList* containing a count of the number of nodes in its two-hop neighborhood using each channel as their fixed channel. The protocol that we simulate balances channel assignments across a two-hop neighborhood. The protocol can easily be extended to balance traffic across a k-hop neighborhood, in which case the

ChannelUsageList will store a count of channels being used in a k-hop neighborhood, and the *NeighborTable* will store a list of neighbors in $(k-1)$-hop neighborhood.

We use two-hop neighborhood information in constructing *ChannelUsageList* since in IEEE 802.11 protocol, when a node A is receiving a packet from a node B on some channel i, all neighbors of B (which are two-hop neighbors of A) are required to not use channel i (this is enforced through the NAV and physical carrier sense mechanisms). Consequently, it is beneficial to ensure that the number of nodes using any given channel as their fixed channel is balanced in a two-hop neighborhood.

Hello packet exchange:

Periodically, each node broadcasts a "Hello" packet on every channel. The hello packet contains the fixed channels being used by the node, and its current *NeighborTable*. When a node receives a hello packet from a neighbor, it updates its *NeighborTable* with the fixed channels of that neighbor. When k-hop balancing is being done, the fixed channels of $(k-1)$-hop neighbors in the received *NeighborTable* are also added to the *NeighborTable*.

The *ChannelUsageList* is also updated using the *NeighborTable* of its neighbor. Updating *ChannelUsageList* with each neighbor's *NeighborTable* ensures that *ChannelUsageList* will contain two-hop (or in the general case, k-hop) channel usage information. Entries which have not been updated for a specified maximum lifetime are removed. This ensures that stale entries, corresponding to nodes that are no longer reachable, are removed from the *NeighborTable* and the *ChannelUsageList*.

The frequency of hello packet exchange depends on the magnitude of average node mobility (in simulations, hello packets are exchanged every 2 seconds). A node moving into a new neighborhood cannot communicate with its neighbors until it has exchanged hello packets with them to learn about the fixed channels being used. Hello packet exchange is used by many routing protocols (such as AODV) as well, and link layer "Hello" information could be merged with messages from routing layers.

Changing fixed channel:

When a new fixed channel is selected, neighbors of the node have to be informed of the change. Therefore, to reduce overheads, we consider changing a fixed channel only at the time when hello

message is due to be sent out. The goal of our algorithm is to move some interfaces from a channel having a lot of fixed interfaces to the channel having the least number of interfaces. Over time, this strategy attempts to ensure that all channels have (approximately) the same number of interfaces. The algorithm used for deciding if the channel assigned to a fixed interface has to be changed is as follows:

1. Compute the number of fixed interfaces on each channel i over all nodes in the k-hop neighborhood, denoted as m_i, using the *ChannelUsageList*. Compute the average number of interfaces per channel in the neighborhood, m_{avg}, and the minimum number of interfaces on any channel in the neighborhood, m_{min}.

2. Among the available fixed channels at the node, the candidate channel that is considered for a potential switch is the fixed channel of the node with the largest value of m_i. Thus, let $m_{fixed} = \max_i(m_i)$ (max operation is only over the fixed channels of the node). If only one fixed channel is available at that node, then that is the candidate channel. This approach ensures that only the fixed interface with the largest count may be switched, and allowing at most one interface to switch per iteration improves the stability of the protocol.

3. If $m_{fixed} \leq m_{avg}$, then a switch is not recommended, as the candidate channel at the node is already well-balanced with respect to the average for the neighborhood. In this case, we are done. Otherwise (i.e., $m_{fixed} > m_{avg}$), we switch the candidate fixed interface, with a probability equal to $\frac{switchProb}{m_{fixed}}$, from the candidate channel to any channel with m_{min} count, provided the new channel is not already a fixed channel of the node. Here, $0 < switchProb \leq 1$ is a protocol parameter used to prevent frequent switching. If all channels with m_{min} count are already fixed channels, then a switch is not done.

With this protocol, on an average (across nodes in the neighborhood), only a fraction $switchProb$ of the m_{fixed} interfaces in the neighborhood switch to a channel with m_{min} interfaces. Using a smaller value of $switchProb$ reduces the number of switches, and thereby reduces oscillations in channel assignments, but requires more time to balance the fixed interfaces across channels. A $switchProb$ of 0.5 was used in the simulations.

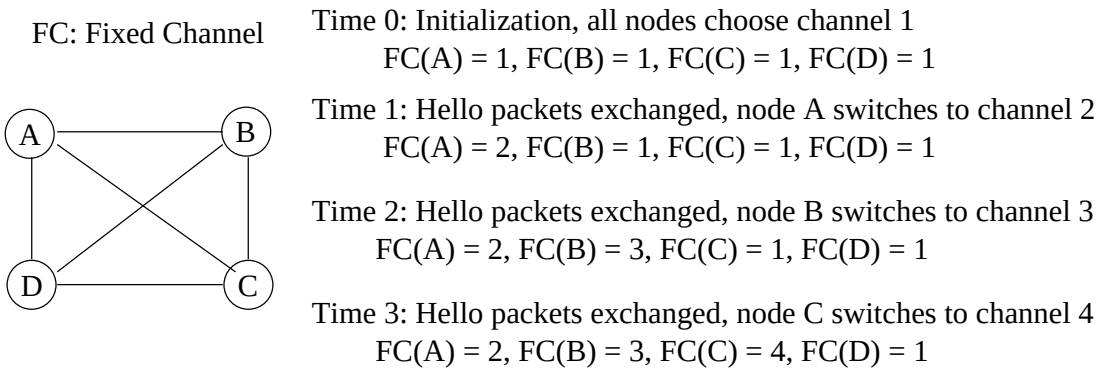

Figure 5.3: Example to illustrate the process of balancing fixed channel assignments. Each node is assumed to have two interfaces, and four channels are available.

Fixed channel selection example:

Figure 5.3 presents a simple example to illustrate the process of balancing fixed channel assignment. In the example, four nodes A, B, C, and D are in communication range of each other. Each node is assumed to have two interfaces, one fixed and one switchable. Four channels are assumed to be available in the network. Initially, let us assume that each node has randomly chosen channel 1 as its fixed channel. We use the notation FC(X) to refer to the fixed channel of node X. After some time, nodes send out a hello message announcing their current choice of fixed channels. At this point, all nodes know that there are four interfaces using channel 1 and zero interfaces on other channels, while the average m_{avg} is only one. Using the balancing protocol, every node randomly decides to switch to a different channel with probability $\frac{0.5}{4} = 0.125$ (assuming that $switchProb$ is equal to 0.5). Let us suppose that only node A succeeds, and switches to channel 2. After the next exchange of hello messages, nodes B, C, and D realize that channel 1 has three interfaces, while channels 3 and 4 have zero interfaces. Therefore, using the balancing protocol, nodes B, C, and D decide to switch with probability $\frac{0.5}{3}$. Let us suppose that only node B succeeds and switches to channel 3. Continuing in this fashion, after some time, all four nodes will have their interfaces on different channels, thereby achieving a balanced assignment.

5.4 Protocol issues

In the previous section, we presented the interface management protocol. In this section, we consider some related issues on how to partition the available interfaces into fixed and switchable interfaces, and how to handle channel heterogeneity.

5.4.1 Selecting the number of fixed interfaces

The interface management protocol, as presented in the previous section, requires the availability of at least one fixed interface and one switchable interface at each node. However, our testbed implementation of the interface management protocol supports nodes with only a single interface as well (more details in Chapter 7). In the testbed implementation, nodes with a single interface use their interface as a fixed interface on one of the channels (recall that, "fixed" means fixed over longer timescales than channel switching and packet transmission times). Such a node can only communicate with other nodes sharing a fixed channel with itself. Broadcast packets are sent out only on the fixed channels, and therefore, single interface nodes have less connectivity compared to multi-interface nodes. The fixed channel is chosen after an initial scan of all the available channels, to choose a channel with at least one multi-interface node. An alternate policy could be to use the channel on which maximum number of multi-interface nodes can be reached (this could improve the connectivity of the node). This protocol extension allows nodes with only one interface to participate in the network, albeit at the cost of reduced connectivity.

When a node in the network has as many interfaces as available channels, then all the interfaces can be fixed. Otherwise, a node that has at least two interfaces (i.e., $m \geq 2$) must partition its interfaces into fixed and switchable interfaces. When there are exactly two interfaces at a node, the interface management protocol requires the first interface to be fixed and the second interface to be switchable. However, when more than two interfaces are available, the interface management protocol can operate with any combination of fixed and switchable interfaces, as long as there is at least one fixed and one switchable interface.

In this thesis, when $m \geq 2$, we use the policy of having $f = \lfloor \frac{m}{2} \rfloor$ interfaces fixed, and the remaining $m - f$ interfaces switchable. With this policy, roughly half the interfaces are fixed, and

the remaining interfaces are switchable. This policy is motivated by the observation that all the data received at a node is over the fixed interfaces, while the switchable interfaces are typically used to transmit data. In the absence of detailed traffic information, it is reasonable to assume that a typical node receives and transmits equal amounts of data. Under that setting, having equal number of fixed and switchable interfaces is reasonable. If a node primarily receives data, it may be beneficial to have more than half the interfaces fixed, and conversely, a node primarily sending data may benefit from having most of the interfaces switchable. In some scenarios, certain nodes may primarily receive or send data (for example, a gateway node may mostly be transmitting down-link data). When such patterns are known, the current implementation of the interface management protocol allows manually specifying the number of fixed interfaces for the desired nodes. However, this process can be automated by adding more intelligence to the interface management module, and the partitioning can be changed dynamically as well.

5.4.2 Dealing with heterogeneity

The interface management protocol has implicitly assumed that all the channels are identical. When channels are not identical, it is possible for transmission ranges on different channels to be different [70, 71]. Variations in range can create unidirectional links with the proposed interface management protocol. For example, suppose that channel 1 has significantly longer range than channel 2. Assume that some node A can communicate with another node B over channel 1, but not over channel 2. Suppose that A uses channel 2 as its fixed channel, while B uses channel 1 as its fixed channel. Then, (broadcast) hello packets from A are received by B (on channel 1), but hello packets from B are not received by A (on channel 2). This creates an unidirectional link from A to B. Unidirectional links can significantly affect the performance of higher layer routing protocols, as node B could assume that it can reach node A (as it has received hello messages from A), while in reality A is not reachable. To avoid the negative aspects of unidirectional links, we have extended the interface management protocol to filter out the unidirectional links. Specifically, a node A does not send data to a node B if node A is not in the neighbor table of B. Node A can check this by looking at the neighbor table of B in the received hello messages from B[3]. Using this

[3]If no hello messages are received from B, then A will anyway not send any data to B.

simple filtering strategy, unidirectional links are eliminated. Smarter higher layer protocols could benefit from using unidirectional links, and better interface assignment (in conjunction with power and rate control) may also reduce the incidence of unidirectional links, but we defer these issues to future work.

When channels are heterogeneous, it is also possible that different channels support a different set of rates. Consequently, the balancing mechanism used in the interface assignment may have to account for the variations in rate as well. Under this setting, it no longer suffices to equally distribute load across channels. Instead, load must be distributed in proportion to data rates supported by channels. However, this extension can easily be incorporated into the balancing policy.

5.5 Performance evaluation

In this section, we evaluate the performance of the interface management protocol. The evaluations in this section assume that routes are specified manually. The next chapter studies the performance of multichannel routing over the interface management protocol.

5.5.1 Simulation parameters

We have simulated the proposed protocols using Qualnet (version 3.9) simulator [72]. Table 5.1 lists the transmit power, carrier sense and receive threshold parameters used to setup qualnet simulations. The ranges shown in the table are the maximum range for each receive threshold (using the specified transmit power). Simulations use a probabilistic bit-error model (as a function of SINR and data rate) to compute error probabilities for packets received within the maximum transmission range. Therefore, the bit-error model allows for success probability to vary across packets sent over a link. The protocols in this thesis were implemented without requiring any modifications to the IEEE 802.11 MAC layer. Unless otherwise stated, the interface switching delay is assumed to be 1 ms. In most figures, we label the curves using the notation (m, c), where m is the number of interfaces per node, and c is the number of channels. All simulations are run for

Transmit power (for all data rates)	20 dBm
Carrier sense threshold	−85 dBm (range of 370 m)
Receive threshold for 6 Mbps data rate	−78 dBm (range of 165 m)
Receive threshold for 9 Mbps data rate	−76 dBm (range of 131 m)
Receive threshold for 12 Mbps data rate	−76 dBm (range of 131 m)
Receive threshold for 18 Mbps data rate	−74 dBm (range of 104 m)
Receive threshold for 24 Mbps data rate	−71 dBm (range of 74 m)
Receive threshold for 36 Mbps data rate	−68 dBm (range of 53 m)
Receive threshold for 48 Mbps data rate	−65 dBm (range of 37 m)
Receive threshold for 54 Mbps data rate	−63 dBm (range of 30 m)

Table 5.1: Default qualnet simulation parameter values.

100 second duration. Each simulation data point is based on 30 runs, and 95% confidence intervals are plotted for each curve.

5.5.2 Chain topologies

We first evaluate the performance of the interface management protocol in simple chain topologies. A chain topology allows us to study the benefits of using multiple channels in a multihop network in the absence of any routing side-effects. 11 nodes are placed to form a chain of 10 hops. The length of the flow on the chain is varied from 1 to 10 hops. We study the performance with both FTP and CBR flows. The flows are always back-logged, i.e., flows continuously send out data at the maximum possible rate so as to saturate the channels. In each experiment, a flow is setup from the first node to the last node of the chain. We set the data rate of all channels to 54 Mbps, the maximum rate possible with IEEE 802.11a. Nodes in a chain are stationary, and direct communication is possible only between adjacent nodes on the chain (distance between adjacent nodes is 30m). This scenario tests the effectiveness of the interface management protocol, and highlights the benefits of using multiple channels.

Figure 5.4 compares the flow throughput in a single channel network, labeled as $(1, 1)$, and the flow throughput in multichannel networks (multichannel networks use the proposed protocol). The throughput of FTP (which uses TCP) and CBR (which uses UDP) flows in a single channel networks rapidly degrades when the number of hops along a chain increases because of two reasons. First, intermediate nodes cannot simultaneously receive and forward data, reducing the achievable throughput by half. Second, since a single channel is used, transmissions on a hop will inhibit other

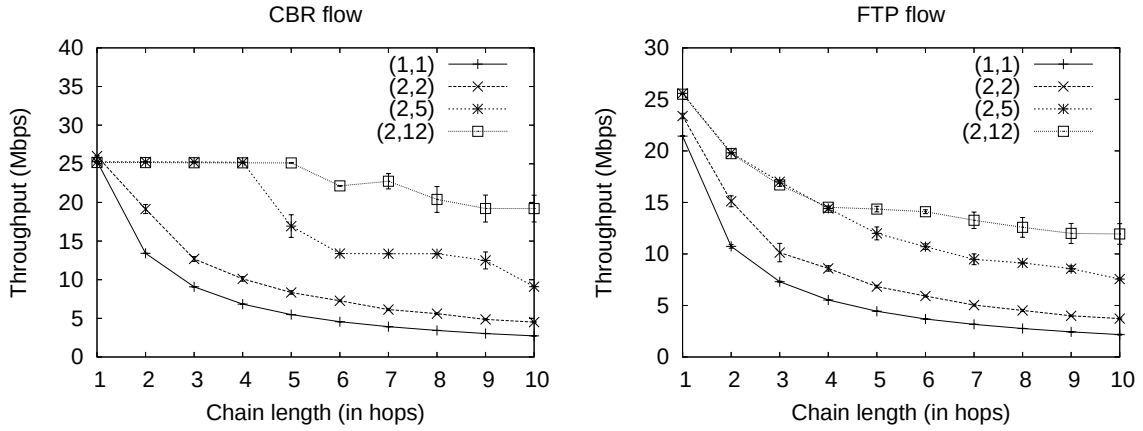

Figure 5.4: Performance of interface management protocol under chain topologies with CBR and FTP traffic.

transmissions on other hops that are within the carrier sense range, thereby further degrading the achievable throughput. Similar results are reported in prior works [73] as well (for single channel networks).

When multiple channels and multiple interfaces are used, the interface management protocol tries to assign the fixed channel of successive nodes along the chain to different channels. Also, when an intermediate node is receiving on the fixed interface, it can simultaneously forward data to the next node using the switchable interface. Consequently, the proposed approach offers higher throughput by using different channels on successive hops, and by using the two interfaces to receive and send data in parallel. From Figure 5.4, we can observe that more channels are useful with longer chains. For example, over a chain of two hops, ideally, only two channels are utilized[4] (one channel for each hop). Therefore, the performance with 5 and 12 channels is nearly the same as the performance with 2 channels over a two hop chain (though higher than that of one channel). On the other hand, over a chain of 10 hops, more channels can be utilized over different hops, and therefore, having 12 channels is better than having 5 channels (and 5 channels is better than 2 channels).

FTP throughput in a chain topology reduces with chain length by a larger factor than CBR

[4]In the figure, having more than two channels helps because when only two channels are available, the channel assignment may sometimes assign the same channel to adjacent hops. When more channels are available, the probability of adjacent hops using the same channel is lower.

throughput, even if sufficient channels are available. FTP uses TCP, and TCP throughput depends on the end-to-end delay. When the length of the chain increases, end-to-end delay increases as well, reducing FTP throughput. This also affects a single channel network, and therefore, having multiple channels still improves performance when compared to a single channel network.

The key conclusion from Figure 5.4 is that multiple channels can significantly improve throughput of a flow in multihop scenarios. Furthermore, even with only a few interfaces (2 in this figure), having large number of channels (up to 12 channels in this example) is beneficial.

5.5.3 Random topologies

We next study the performance of the interface management protocol in static random network topologies. Random topologies are created by placing 50 nodes uniformly at random in an area of 500 m by 500 m. Each node sets up a back-logged CBR or FTP flow to a randomly chosen neighbor. The transmission data rate is assumed to be 12 Mbps, and for the parameters used in the simulation, the maximum transmission range for this data rate is approximately 130 m. In Figure 5.5, we compare the aggregate network throughput for 10 random topologies for different number of channels and interfaces. All throughput values are normalized with respect to the throughput in the single channel, single interface scenario. We first normalize the aggregate throughput for each of the 30 runs, and the figures plot the average normalized throughput (over 30 seed values) and the confidence intervals of the normalized throughput. Therefore, the throughput curves represent the "factor" by which a multichannel multi-interface network outperforms a single channel network. The topologies are sorted by the performance improvement in a $(2, 12)$-network with CBR flows.

As we can see from the figure, there is a significant performance improvement when multiple channels are available. The magnitude of performance improvement is dependent on the number of channels that are available. Furthermore, the magnitude of improvement is higher for CBR traffic, in comparison with FTP traffic. FTP flows use TCP, and it is well-known that TCP throughput is inversely proportional to the round trip time of the route. The higher bandwidth available by using multiple channels reduces the packet transmission time component of the end-to-end delay. However, end-to-end delay has other fixed components, such as interface switching delays, which are independent of the number of channels. Therefore, the overall end-to-end delay does not reduce

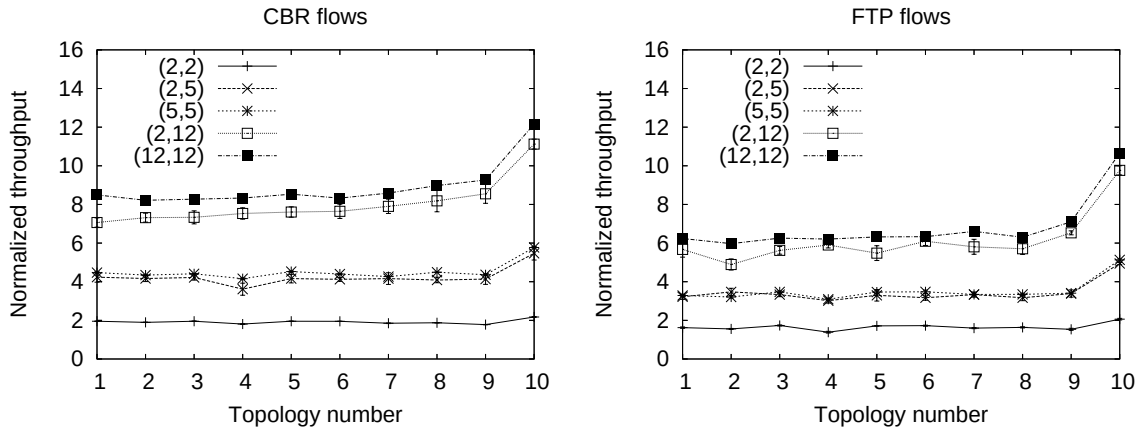

Figure 5.5: Performance of interface management protocol under random topologies with CBR and FTP traffic.

in proportion to the increase in the flow bandwidth. Consequently, TCP throughput does not improve to the extent seen with CBR traffic (which uses UDP).

Another observation from Figure 5.5 is that even having only two interfaces per node is sufficient to achieve most of the benefits of having large number of channels. For example, a $(2, 12)$-network achieves nearly the same throughput as a $(12, 12)$-network (the difference is less than 10% in most cases). This confirms our analysis that with a moderately dense network, it is possible to exploit a large number of channels with a few interfaces.

Figure 5.6 studies the performance of interface management protocol for two other node densities, with FTP traffic. In the figure, we consider random topologies having 25 nodes and 75 nodes, and the nodes in both cases are distributed in an area of 500 m by 500 m. As we can see from the figure, the magnitude of improvement is sometimes higher for the denser network. When the network density is less, some parts of the network may not have sufficient interfaces, reducing the magnitude of improvement. Therefore, we can see that in the lower density scenario, having more than two interfaces per node helps to improve performance (by as much as 25% in some topologies when 12 interfaces are available). In a denser network, there are a sufficient number of interfaces in the neighborhood even if each node has only two interfaces. Therefore, having more than two interfaces does not improve performance significantly.

Figure 5.7 studies the network load imposed by the "hello" messages transmitted by the interface

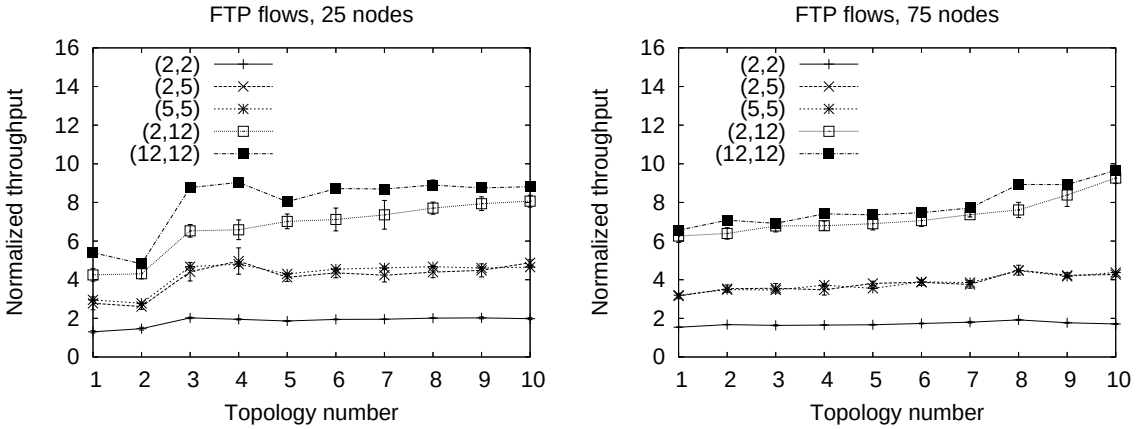

Figure 5.6: Performance of interface management protocol under random topologies with varying network density.

management protocol. The overhead is measured over 50 node and 25 node random topologies (the same topologies used for measuring throughput). As we can see from the figure, the hello message overhead (plotted in kilo bytes per second) consumes only a small fraction of the network capacity. We also observe that the hello message overhead increases both with the number of channels and the network density.

Each node sends out a hello message periodically, and the hello overhead is directly proportional to the frequency of sending out hello messages[5]. Furthermore, since hello messages are broadcast, a copy of the hello message has to be sent out on every channel. Therefore, the hello message overhead (measured as total bytes sent over all channels) also increases linearly with the number of channels. However, if the overhead is measured on a per-channel basis, then the overhead will be independent of the number of channels. The network overhead is measured by adding up the hello message overhead over all nodes in the network. Therefore, the overhead also depends on the network density, and increases linearly with the number of nodes in the network.

5.6 Discussions

The interface management protocol is capable of utilizing all the available channels without requiring tight synchronization between nodes. The protocol distributes traffic across channels by

[5]In our simulations, we use a hello message frequency of two seconds.

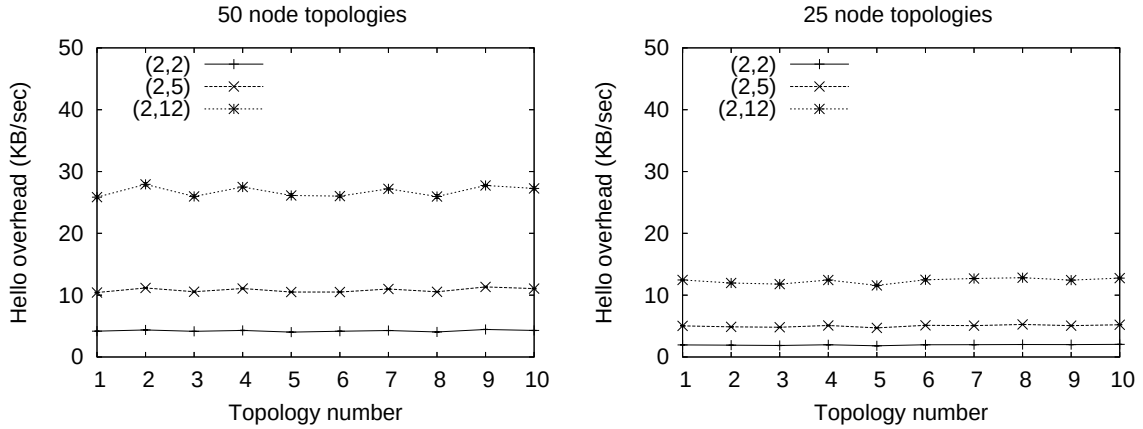

Figure 5.7: Overhead of hello message exchange. The overhead depends on number of channels, and node density.

carefully balancing the number of interfaces assigned to a channel. The protocol is designed to scale when more channels and interfaces become available. Simulation results have shown that the interface management protocol is capable of achieving significant performance improvements when multiple channels are available, even if each node is equipped with only two interfaces.

The algorithm for balancing interfaces across channels is traffic agnostic. In specific multihop scenarios where traffic information is available, and does not change frequently (such as in mesh networks), it may be better to balance channel assignment using traffic information. Our choice of a hybrid interface assignment strategy is based on two factors: the difficulty of achieving tight co-ordination between nodes, and switching delay being non-negligible. If both these assumptions are invalid, then alternate interface assignment strategies may turn out to be more suitable of multichannel networks, and this is an interesting avenue for future work.

CHAPTER 6

MULTICHANNEL ROUTING PROTOCOL

In this chapter, we motivate the need for a new routing metric, and describe the proposed multichannel routing (MCR) metric. We then briefly describe the integration of the metric into an on-demand multichannel routing protocol. The routing protocol operates in conjunction with the interface management protocol. We conclude the chapter with detailed evaluations of the combined interface management and routing protocols.

6.1 Motivation for designing a new routing metric

Any existing routing protocol can potentially be used over the previously described interface management protocol (since the interface management protocol transparently manages multiple channels and interfaces). The interface management protocol ensures that within a neighborhood, different nodes use different channels to the extent possible. However, if a multihop route is not carefully selected, successive hops may not use different channels, thereby not fully utilizing channel diversity.

Some routing protocols, such as AODV and DSR, use the shortest-path metric for route selection. Shortest-path metric assigns an unit cost for each hop in a route, and does not distinguish between a route that uses many channels, and a route that uses few channels. Figure 6.1 illustrates the need for choosing channel diverse routes. Suppose that all nodes have already chosen their fixed channels as shown in the figure. Assume that node A wants to find a route to node C. Node A has two route choices: A-B-C or A-D-E-C. Route A-B-C requires only two hops, but both the hops share a common channel (channel 3), reducing the end-to-end throughput to half the link throughput. Although route A-D-E-C is longer, it uses different channels on each hop and can

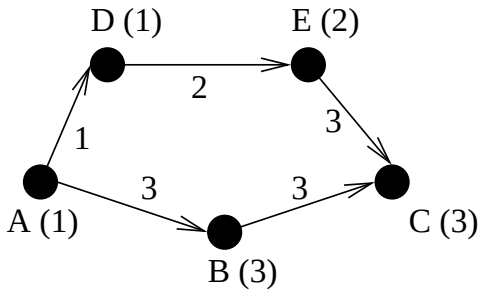

Figure 6.1: Need for selecting channel diverse routes. Each node is labeled with its fixed channel. Each link is labeled with the channel used on that link.

potentially support a higher end-to-end throughput (equal to the link throughput). Therefore, for this example, a shortest hop routing metric would choose the sub-optimal route. In this chapter, we present a new multichannel routing metric (MCR) that selects channel diverse routes when such routes provide higher throughput, even if the routes are longer.

Since interfaces may have to be switched between channels, the expected time it takes a packet to traverse a link depends on whether interface switching is required before packet transmission or not. Multi-radio link quality source routing (MR-LQSR) [4] is a routing protocol proposed for the scenario where there are as many interfaces per node as channels. However, the metric used by MR-LQSR, called WCETT, can be extended for use over our interface management protocol. In the rest of this section, we designate the WCETT version adapted for use over the interface management protocol as "m-WCETT" (modified WCETT). The m-WCETT metric is capable of selecting channel diverse paths. However, m-WCETT was designed for the scenario where interfaces are permanently fixed on specified channels. In our architecture, switchable interfaces have to switch among multiple channels, and the routing metric has to account for switching cost as well.

Figure 6.2 illustrates the need to account for the delay incurred in switching. Assume that node B is already routing data to node E. Suppose that node A is setting up a route to node C, with two possible routes: A-B-C, and A-D-C. Both routes A-B-C, and A-D-C use the same set of channels, and hence have the same channel diversity. However, if route A-B-C is chosen, node B has to frequently switch between channels 2 and 3 when sending data to node E and node C respectively. Such frequent switching may impose a significant overhead, and the throughput over both flow A-B-C and flow B-E may reduce. The MCR metric that we present modifies the

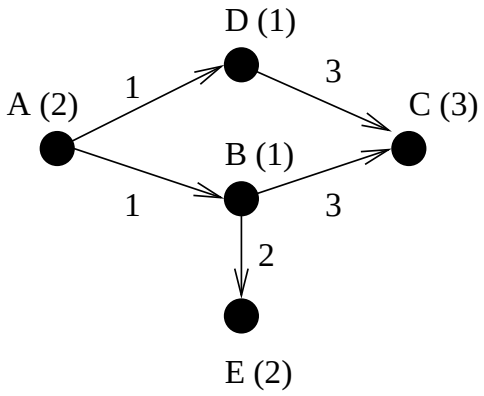

Figure 6.2: Need for considering interface switching cost. Each node is labeled with its fixed channel. Each link is labeled with the channel used on that link.

m-WCETT metric to incorporate switching cost.

In the next section, we first describe our approach for measuring switching cost. The interface management protocol is extended to compute the switching cost, and this cost information is used by the routing protocol. We then revisit a link[1] cost metric (called ETT) used in WCETT, and highlight the modifications needed for the scenario where the number of interfaces per host is smaller than the number of channels. Individual link costs and switching costs are then combined into a new path metric (MCR) that is used by the routing protocol.

6.2 Proposed MCR metric

In this section, we propose a new multichannel routing metric. The metric requires measurement of the interface switching cost, and the link cost, and these costs are combined into a path cost.

6.2.1 Measuring interface switching cost

The interface switching cost of a channel should measure the increase in delay a packet experiences on account of interface switching. Since a packet sent out on a fixed interface is never delayed waiting for the interface to switch, we set the switching cost of a fixed channel to be zero.

[1] A link refers to a pair of nodes and a specific channel used for communicating between the nodes. When there are c channels, each pair of nodes can have up to c links.

Frequent interface switching is necessitated if a node has large amount of data to send on more than one switchable channel. For identifying such scenarios, on every switchable channel j, we measure the likelihood that the switchable interface is on a different channel when a packet has to be sent out on channel j. To estimate this, we maintain a variable, called *InterfaceUsage(j)*, for each channel j to measure what fraction of time a switchable interface was transmitting on channel j. If during some time interval a switchable interface is tuned to a channel j, but is idle, then that time is not included as part of *InterfaceUsage(j)*, because the idle time could be used later to transmit data on some other channel. Interface usage of each channel is maintained as an exponentially weighted average over one second intervals. Therefore, the sum of *InterfaceUsage* values over all channels is less than or equal to 1 second.

If a route chooses to use some channel j, then we estimate the probability $p_s(j)$ that the switchable interface will be on a different channel ($i \neq j$) when a packet arrives on channel j to be:

$$p_s(j) = \sum_{\forall i \neq j} InterfaceUsage(i) \tag{6.1}$$

Note that $p_s(j)$ computation assumes that all the currently available interface idle time can potentially be used later on channel j. We can directly sum up *InterfaceUsage()* values to obtain a valid probability because the usage time is measured over one second intervals. If the measurement window duration is changed, then the interface usage values will have to be divided by the measurement window duration to obtain a valid probability. In addition, if multiple switchable interfaces are available, then the usage values will have to be divided by the number of interfaces to obtain a valid probability. Next, the switching cost of using channel j is measured as:

$$SC(j) = p_s(j) * switchingDelay \tag{6.2}$$

where *switchingDelay* is the interface switching latency, which can be estimated offline. With this definition, switching cost may be viewed as the *expected increase in the latency* incurred by a packet because of interface switching.

Considering switching cost during route selection ensures paths that require frequent switching

are not preferred. When some flow is already passing through a node and using some channel j, then the switching cost of all other channels (except the fixed channel) will be high. Hence, new routes through the node using any other channel will have a higher cost, and therefore will not be preferred. However, this strategy does not reduce network connectivity because if a single route is available between a pair of nodes, then that route is selected even if its cost is high.

6.2.2 Measuring individual link costs

The cost of a link is measured as the expected transmission time (ETT) required to transmit a packet over the link. ETT [4] of a link (between a pair of nodes on some channel j) is defined to be:

$$ETT = ETX * \frac{S}{B} \tag{6.3}$$

where ETX [67] is the expected number of transmission attempts (including retransmissions) required for transmitting a packet (equation 6.5, described later), S is the average packet size, and B is the data rate of the link. When "autorate" feature is enabled, or a combination of IEEE 802.11a and IEEE 802.11b hardware is used, transmissions from a node to different neighbors may use different data rates. The link data rate can be measured using probe packets [4], or can be read by querying the driver (this support is available with newer hardware). In our simulations, we assume that the link data rate is probed from the driver.

Measuring ETX: The technique used to compute ETX (expected transmission count) is based on a modification of the technique presented in [4]. The ETX of a link from a node X to a node Y on some channel j depends on the forward packet loss probability from X to Y on channel j (p_f), and the reverse packet loss probability from Y to X on channel j (p_r). Assuming that the IEEE 802.11 MAC is used, a data transmission is successful only when the packet is successfully acknowledged. Consequently, the probability that a transmission along the link fails is given by,

$$p = 1 - (1 - p_f) * (1 - p_r) \tag{6.4}$$

Once the link failure probability p is computed, then the expected number of transmissions

(ETX) needed before a transmission is successful is given by,

$$ETX = \frac{1}{1-p} \tag{6.5}$$

Note that the ETX computation is not completely accurate because IEEE 802.11 MAC allows only a fixed number of retransmission attempts (while Equation 6.5 assumes that infinite number of retransmissions are possible). However, in most scenarios, the link failure probability p is not too large, and ETX measurement is accurate for those scenarios.

In [4], explicit probe packets are broadcast by a node for measuring the loss rate. To reduce overheads, we use the "Hello" packets that are anyway broadcast by the link layer protocol as probe packets. The forward and reverse loss probabilities were measured in [4] using the following approach. A node X measures the loss rate to Y on some channel j by measuring the probe packet loss rate. Similarly, Y measures the loss rate from X on channel j as well, and informs X of the measured loss rate. Thus, both forward and reverse path probabilities between any pair of nodes can be estimated on every channel (since in [4], each node has an interface on every channel).

However, under our problem scenario, it is difficult to measure the loss rate between two nodes X and Y on *all channels*, using the above approach. Note that we assume that the number of interfaces may be smaller than the number of channels, and hence a node will not have an interface listening on every channel. Furthermore, we only need to measure the loss rate on the channel that will be actually used for communication. When using the previously described link layer protocol, a node X can measure the packet loss probability from a neighbor Y only on its own *fixed channel*, as that is the only channel on which the node X is always listening and can correctly count the number of packets sent by Y. During route discovery, when a node Y receives a route request packet from a node X on Y's fixed channel j, the forward loss probability from X to Y on channel j is known (based on Y's earlier measurements on channel j), but the reverse loss probability from Y to X is not known (as X may be using some other channel $k \neq j$ as the fixed channel).

Hence, for computing ETX, we make the simplifying assumption that *the reverse loss probability is equal to the forward loss probability*, i.e., $p_r = p_f$, though this assumption may not always hold in practice (because of asymmetries arising out of interference and channel fading). Our simulation

model allows for asymmetric packet losses, and therefore the impact of the simplifying assumption has been accounted for in the simulations.

6.2.3 MCR: The path metric

The proposed multi-channel routing (MCR) metric combines the measured link ETT and switching costs into a single path cost. Both the ETT cost and the switching cost are measured in units of time, and therefore can be meaningfully combined into a single metric. The key goal of the path metric is to identify the "best" route. However, the notion of "best" route depends on the type of traffic that is using the route. For example, UDP throughput is primarily limited by the bottleneck bandwidth. If we assume that all links along a route that use a common channel interfere with each other[2], then the capacity of a link is inversely proportional to the number of links sharing that channel. Under this model, the route will be bottlenecked by the channel having the maximum number of links. Therefore, one choice for a path metric is to minimize the maximum number of links on any channel.

TCP throughput is dependent on the end-to-end delay (and the packet loss probability). End-to-end delay depends on the total number of hops along a path, and the number of interface switches, in addition to the bottleneck bandwidth. Therefore, for TCP traffic, the total number of links should also be considered in the path metric. Furthermore, when a route is selected, consideration should be given to other flows in the network as well. The bandwidth resources consumed by a flow is proportional to the number of hops in the network, while the interface switching cost is proportional to the number of hops on which interfaces have to be switched. Therefore, for maximizing the aggregate network throughput, it is beneficial to minimize the number of hops and the number of interface switches, especially when there are a large number of flows in the network.

We balance the requirements of minimizing the maximum number of links on any particular channel, while also minimizing the number of hops and number of switches, by combining the two components into a metric that is a weighted combination of the two objectives. This weighted

[2]This assumption is pessimistic for long routes, but is fairly accurate as long as there are at most 5 to 6 hops on a route.

combination approach is based on the technique proposed by [4] for WCETT metric. Next, we describe the proposed MCR metric.

The ETT cost of the i^{th} hop of a path is designated as ETT_i. The switching cost for the i^{th} hop is given by $SC(c_i)$, where c_i is the channel used on the i^{th} hop. The total ETT cost on any channel j, X_j, is defined as:

$$X_j = \sum_{\forall i, \text{such that } c_i = j} ETT_i \tag{6.6}$$

When the routes are expected to be long, the above definition can be modified to measure the cost over only those hops that interfere with each other. In that setting, X_j will be defined as the maximum of sum of ETT values over any set of interfering hops on channel j. Using the above definitions, the MCR metric, which measures the path cost, is defined as:

$$\text{MCR} = (1-\beta) * \sum_{i=1}^{h} \left(ETT_i + SC\left(c_i\right)\right) + \beta * \max_{1 \le j \le c} X_j \tag{6.7}$$

where β is a weight between 0 and 1, h is the number of hops on the path, and total number of available channels is c.

The MCR metric is a weighted combination of two components, similar to WCETT [4]. The first component measures the sum of ETT and switching cost values along the path, and may be viewed as measuring the "resource" consumed along the path. The first component is aimed at minimizing the number of hops used and the number of interface switches required along a path. The second component measures the cost of the "bottleneck channel" along the path[3]. Since transmissions along different channels do not interfere, the path throughput is constrained by the throughput along the more heavily used channel. The cost of second component will be small if a channel diverse path is used. Thus, the second component ensures that channel diverse paths are selected, while the first component ensures that adding more hops or switches to a path increases its cost. Simulation results with various β suggest that β should not be close to either 0 or 1, and

[3]The second component does not include switching cost because fraction of channel bandwidth available to a node is not affected by switching incurred by the nodes

throughput is fairly insensitive to values in between. Therefore, we choose a value of 0.5 in our simulations.

6.2.4 Properties of MCR metric

The MCR metric is designed to select channel diverse routes while ensuring that the length of paths is not significantly larger than the shortest path. When there is a single flow in the network, we have already argued that the MCR metric can provide good performance with both TCP and UDP traffic. We next consider the performance when there are a large number of flows by considering the asymptotic setting used in the capacity analysis.

In the capacity analysis, it was shown that straight line routing, in conjunction with careful interface assignment, is optimal. Shortest line routing is asymptotically equivalent to shortest hop routing, and the MCR metric approaches shortest hop routing in a large network. Note that with the traffic model used in capacity analysis, even the shortest route will have roughly $O\left(\sqrt{\frac{n}{\log n}}\right)$ hops (with up to $O(\log n)$ channels). When routes have a large number of hops, the dominant cost in the routing metric will be the first component, and the routes chosen using MCR will be close to the shortest path. Within the space of routes that are close to the shortest hop route, the MCR metric will try to maximize the channel diversity. Furthermore, because the metric includes switching costs, it will generally prefer nodes forwarding fewer flows (because such nodes generally require less frequent switching). Because of these reasons, MCR will approximate optimal routing in the asymptotic setting.

Insights from capacity analysis suggest that flow balancing is required in multichannel networks. In the MCR metric design, we have not explicitly included flow balancing for simplicity. The flows in the capacity analysis are long-lived (because the set of flows is fixed), while in practice, there may be a mix of short-lived and long-lived flows. If there are a large number of short-lived flows, then the metric cost may oscillate if flows are accounted for in metric design. Therefore, we have not currently included flow balancing, but the metric can be easily extended to include flow balancing costs. Nevertheless, the switching cost component implicitly tries to balance flows across switchable interfaces (because a switchable interface forwarding more traffic may have a higher switching cost).

However, the switching cost is zero if a new flow uses the same channel (to forward data) as all the existing flows that use the switchable interface, or if the new flow uses the fixed interface. Therefore, the switching cost does not balance the flows in all scenarios. We defer to future work extension of the MCR metric to support explicit flow balancing.

One desirable property of a routing metric is "isotonicity" [74]. Consider two routes from a node A to a node B, say R1 and R2. Suppose that the cost of route R1 is less than the cost of route R2. Then, an isotonic metric ensures that cost of a route from A to some node C, through node B, is smaller for a route containing R1 than a route containing R2. Isotonic metrics allow the use of efficient (greedy) algorithms, such as Bellman-Ford, to find optimal routes. In the absence of isotonicity, finding the optimal route may require examining all possible routes, which is quite expensive. However, the proposed MCR metric is not isotonic (and neither is WCETT or m-WCETT). For example, consider one sub-route from A to B having one hop on channel 1 and one hop on channel 2, and another sub-route from A to B having two hops on channel 1. In this case, the cost of second sub-route is higher. Now, suppose that a route is found from B to C that uses two hops on channel 2. Then, the combined route from A to C using the first sub-route has higher cost than the combined route from A to C using the second sub-route. This example shows that the MCR metric is not isotonic.

The MCR metric is integrated into an on-demand routing protocol. Because the metric is not isotonic, the selected route may not always be optimal. However, we use simple mechanisms (described below), to improve the probability of finding good routes. It is also possible to convert the MCR metric into an isotonic metric using the techniques proposed in [74].

6.3 Route discovery and maintenance

The MCR metric is incorporated into a source-initiated on-demand routing protocol, similar to DSR [62]. The route discovery process is initiated by a source node, which broadcasts a Route Request (RREQ) packet over all channels[4]. Each new route discovery initiated by a node uses an unique sequence number which is included in all RREQ packets. The RREQ packet sent by a node

[4]Overhead can be reduced by initially sending requests over a subset of channels only, and later sending requests over all channels if no route was discovered during the initial attempt.

X over a channel i contains a list of the link costs, the switching costs, and channels used, for all previous hops, as well as the switching cost of channel i at node X. On receiving a RREQ, a node can compute the ETT of the previous hop based on the previously measured link loss rate. An intermediate node re-broadcasts the RREQ in the following two cases:

1. The sequence number in the RREQ is being seen for the first time. In this case, the cost of the already traversed path is stored in a local table.

2. The cost of the already discovered path in the RREQ is smaller, or larger by at most a threshold percentage, than the cost seen in all earlier RREQs with the same sequence number (if any).

A lower cost RREQ may traverse a longer path, and reach an intermediate node after a higher cost RREQ has been forwarded. In addition, as we discussed earlier, the MCR metric is not isotonic. Consequently, even a partially discovered route with a slightly higher cost may eventually lead to a lower cost route to the destination. Therefore, the second condition, not present in DSR, is required to improve the probability of discovering the least cost route. By increasing the threshold percentage, we can increase the probability of discovering the least cost route, but at the cost of an increase in the routing overhead. Our simulations have used a threshold percentage of 5%, and we have observed that small values of the threshold percentage are sufficient to find good routes.

When the destination receives a RREQ, it responds with a route reply (RREP) only if the cost of the received RREQ is smaller than other RREQs (containing the same sequence number) seen till then. We use a procedure called "Route Refresh" (not present in DSR), wherein, a new route discovery is periodically initiated (the period is set to 20 seconds in simulations) to update the costs of known routes, even if they are not broken. This mechanism ensures that the route cost information is never too old, and new lower cost routes, if any, are discovered.

The routing protocol retains other features of DSR, such as route repair using route error messages. However, we have not used optimizations such as route caches and packet salvaging, though it may be possible to extend the protocol with those optimizations.

6.4 Performance evaluation

In this section, we evaluate the performance of the proposed interface management and routing protocols through qualnet simulations. The parameters used to set up qualnet simulations were listed in Table 5.1 (in Chapter 5). The parameters for the interface management protocol are the default values specified in Chapter 5. Unless otherwise stated, simulations are over topologies having 50 nodes that are placed uniformly at random in a 500 m by 500 m area. All nodes are stationary. Simulations are run for 100 seconds. Each simulation data point is based on 30 runs, and 95% confidence intervals are plotted for each curve.

We compare the performance between networks having 1, 2, 5 and 12 channels. Most curves are labeled using the notation (m, c), which represents a network having m interfaces and c channels. Single channel scenario is evaluated with the DSR[5] routing protocol (to estimate baseline performance numbers). Unless otherwise stated, multichannel scenarios are evaluated with the proposed MCR metric. The interface switching delay is assumed to be 1 ms. Since the aggregate throughput obtained depends on the topology, we normalize all results with the throughput obtained when using DSR on a single channel, to allow comparison across topologies. We first normalize the aggregate throughput for each of the 30 runs, and the figures plot the average normalized throughput (over 30 runs), and the confidence intervals of the normalized throughput. The normalized throughput quantifies the "factor" of performance improvement of the proposed multichannel protocol with respect to a single channel network.

6.4.1 Performance in random topologies

Figure 6.3 compares the throughput of single channel DSR with multichannel MCR when using FTP (which uses TCP) and CBR (which uses UDP) traffic. All flows are always back-logged. Each of the 50 nodes in the network sets up a flow to a randomly selected destination. This setup simulates the traffic pattern studied in the capacity analysis. Results are plotted for 10 different random topologies. The topologies are numbered from 1 to 10 in the increasing order of the normalized throughput in scenarios with 12 channels and using CBR traffic. The results clearly

[5]We use the a version of DSR [62] with caching and other optimizations enabled. A destination node sends a reply for all route requests, and a source node uses a route with fewest hops.

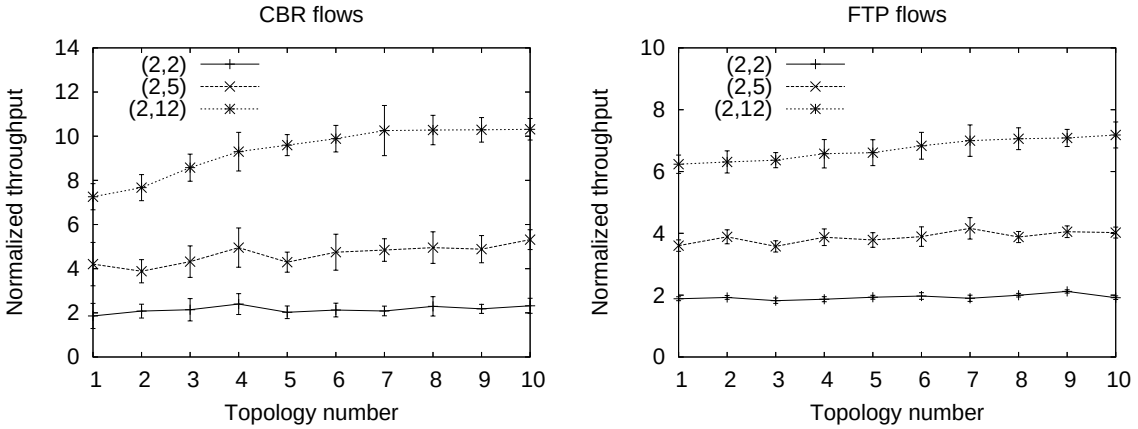

Figure 6.3: Throughput in random topologies. Every node sets up a flow to a randomly chosen destination node.

show that even if only two interfaces are available, more than a two-fold improvement is possible. In fact, the magnitude of improvement is proportional to the number of channels, even when only two interfaces are available.

As we can see from the figure, in general, higher performance improvement is seen in the case of CBR traffic. For example, the throughput improvement of MCR with 12 channels under FTP traffic varies from a factor of 5.7 to a factor of 7.6, depending on the topology. With CBR traffic, the throughput improvement of MCR with 12 channels varies from a factor of 7.7 to a factor of 12.4. The performance improvement with 12 channels is sometimes larger than a factor of 12 because distributing traffic across channels decreases MAC contention overhead. The performance improvement with FTP (which uses TCP) is smaller because TCP performance depends both on the available bandwidth and the end-to-end delay. Since the end-to-end delay does not reduce by a factor proportional to the number of channels, throughput improvement with FTP is lower.

The magnitude of improvement obtained with MCR depends on the underlying topology and traffic patterns. If multiple routes are available between a pair of nodes, then MCR will choose a good *channel diverse* route, resulting in significant throughput improvement. Otherwise, MCR may be forced to use a less channel diverse route, resulting in lower throughput improvement. In general, the results suggest that MCR *can offer significant improvements even when using only two interfaces*. In contrast, the results for a $(2, 2)$-network corresponds to the throughput obtained by

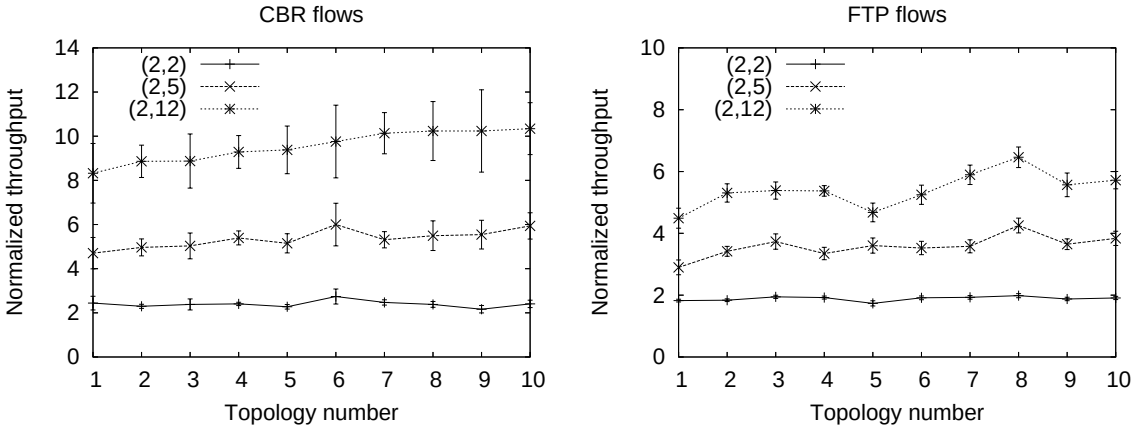

Figure 6.4: Throughput in random topologies with low node density.

MR-LQSR protocol [4], since MR-LQSR uses only two channels when two interfaces are available. When more than two channels are available, say five, MR-LQSR only achieves the throughput of the $(2, 2)$ curve, while our solution achieves the throughput of the $(2, 5)$ curve. It is clear from this result that when more channels are available, our approach performs significantly better than past works that used only as many channels as interfaces.

Figure 6.4 plots the throughput improvement in a low density network having 25 nodes distributed in a 500 m by 500 m area. As before, each node sets up a flow to a random destination. Similarly, Figure 6.5 plots the throughput improvement in a high density network having 75 nodes distributed in a 500 m by 500 m area. As we can see from the figures, the performance improvement is sometimes higher in networks with higher density, although the performance improvement does not significantly vary with node density.

We also evaluate the throughput improvement when there are a smaller number of flows in the network. We vary the number of flows in the network from 1 to 10. The throughput for a given number of flows is obtained by averaging over different random topologies with 50 nodes (the same topologies used above). Figure 6.6 compares the normalized throughput with varying number of flows. When the number of flows in the network is small, throughput does not significantly increase for larger number of channels. As the number of flows increase, using larger number of channels provides significantly better performance than using a single channel.

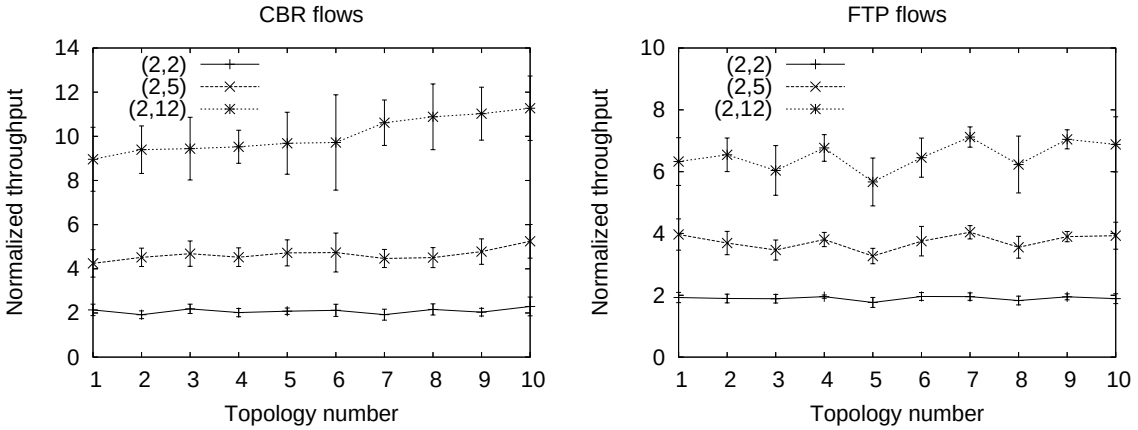

Figure 6.5: Throughput in random topologies with high node density.

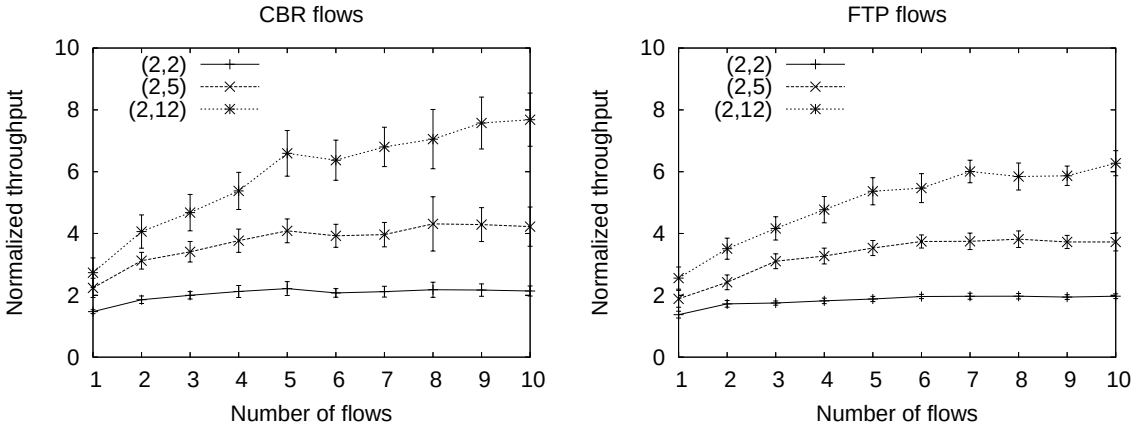

Figure 6.6: Throughput with varying number of flows. Results for a flow are averaged over 10 different random topologies.

When the number of flows is small, the throughput improvement depends on the channel diversity available on the best route between the source and the destination. Note that a single flow can use only as many channels as the number of hops on the flow that interfere with each other. As a result, the full benefits of using a large number of channels (say, 12) is not realized when the number of flows is small. When the number of flows is large, hops of different flows that share an interference neighborhood can potentially use different channels, thereby improving the channel utilization. Therefore, when the number of flows is large, at any point of time, it is likely that every neighborhood has some subset of flows that can utilize the available channel diversity. Furthermore, increasing the number of flows in the network increases the average contention at the MAC layer. When multiple channels are available, the fixed channels of various nodes are distributed across the available channels. Since the number of nodes using a specific channel decreases, MAC layer contention on each channel reduces (by a factor proportional to the the number of channels). Therefore, by using all the available channels, MCR can provide better scalability with increasing network contention, than a single channel solution.

Figure 6.7 plots the total routing overhead with varying number of flows (same setup as used in Figure 6.6, with FTP traffic). As expected, the routing overhead increases linearly with the number of active flows in the network. Since route request and route refresh messages are broadcast (route reply and route error messages are unicast), a copy of those messages have to be sent out on every channel. Therefore, the routing overhead also increases with the number of channels. Nevertheless, the total routing overhead is fairly small compared to the total network capacity.

We next evaluate the impact of the β parameter used in the MCR metric (see Section 6.2.3) on the network performance. With a small value of β, the MCR metric gives less importance to channel diversity, while with a large value of β, less importance is given to minimizing resource consumption in the network. Figure 6.8 compares the throughput improvement with different values of β, under CBR and FTP traffic, as the number of flows in the network is varied from 1 to 10. All throughput values are normalized with respect to throughput with $\beta = 0.5$, for ease of comparison. The key observation from the figure is that the protocol performance is not very sensitive to β around 0.5. There are some scenarios where using a β of 0.7 or 0.3 reduces throughput, but in general, the performance is good for β equal to 0.7 or 0.3. However, when β is close to 0 or 1, then

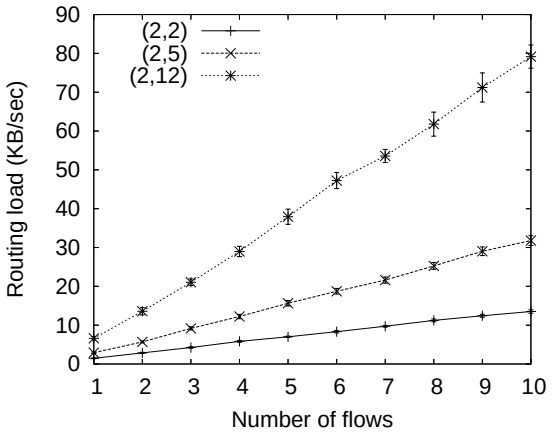

Figure 6.7: Overhead imposed by the routing protocol.

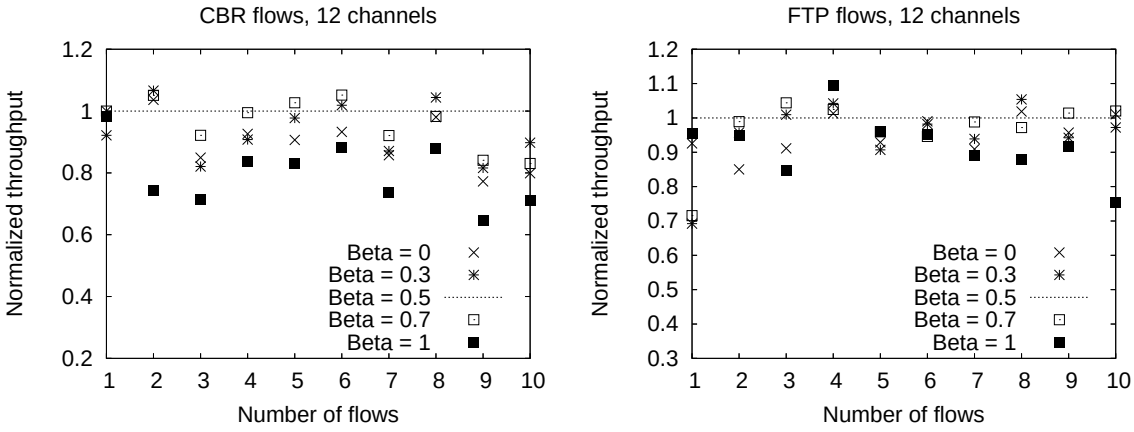

Figure 6.8: Impact of routing metric parameter β.

the performance is worse for most scenarios. Therefore, other than β close to 0 or 1, the protocol performance is not very sensitive to the exact value of β. All our other simulations in this chapter use a β of 0.5, which appears to be a reasonable choice.

6.4.2 Impact of switching delay

A design goal of this thesis is to develop protocols that are useful over a range of interface switching delays. Most of the simulations use an interface switching delay of 1 ms (which we argued to be a reasonable choice in Chapter 2.1). Some of the commercially available hardware today have a switching delay of 5 ms, while future hardware may reduce switching delays to as low as 0.1 ms

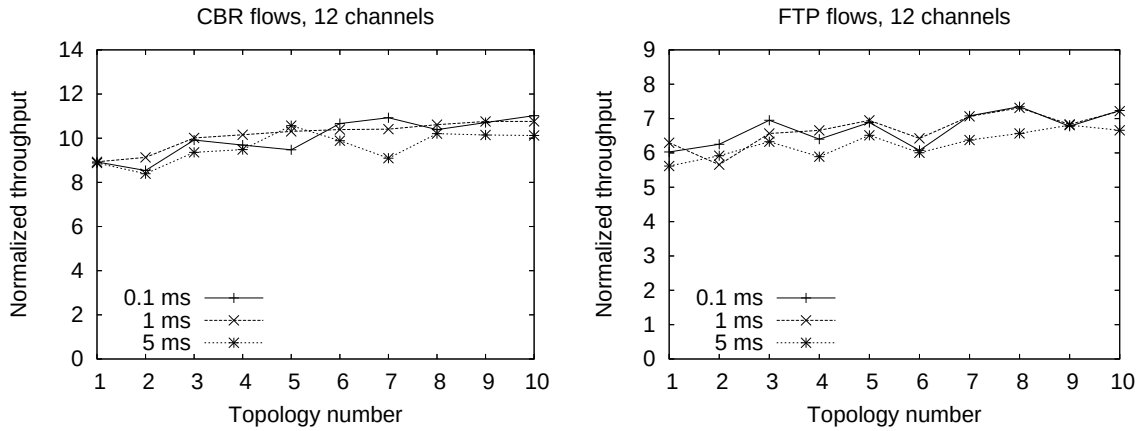

Figure 6.9: Impact of switching delay on throughput.

(or even lower). We next evaluate the performance of the proposed multichannel protocol over a range of switching delays.

Figure 6.9 compares the throughput of the proposed protocol in a $(2, 12)$-network with FTP traffic under random network topologies (same topologies and traffic as used in Figure 6.3). Confidence intervals have not been shown for the results here because the intervals overlap, reducing clarity (the standard deviations for all data points are less than 0.35). As we can see from the figure, variations in switching delay does not have a significant impact on network performance. With our protocol design, a larger switching delay does not necessarily degrade throughput. The routing metric accounts for the cost of interface switching, and attempts to reduce the number of interface switches. Any remaining switches beyond that has only a small impact on performance. For example, when some node is switching its interface, other nodes in its vicinity could still be using all the channels. Therefore, although larger switching delay reduces the utilization of switchable interfaces, it may not reduce the utilization of channels. In addition, the *maxSwitchTime* parameter of the link layer protocol ensures that when possible, the time spent on a channel is several times larger than the time spent in switching. The combination of these effects ensures good performance even with moderately large switching delays.

6.4.3 Comparison of metrics

Different routing metrics can be used in conjunction with our interface management protocol. However, using illustrative examples, we have shown that shortest hop metric is not suitable in a multichannel network. In addition, we have shown that the multichannel metric m-WCETT is not always sufficient in a multichannel network where interfaces switch, and we extended m-WCETT to obtain the proposed MCR metric. We now compare the performance of shortest hop metric[6] and m-WCETT with the proposed MCR metric.

Figure 6.10 compares the throughput (over 30 scenarios) obtained by different metrics in a $(2, 12)$-network with both FTP and CBR traffic. The number of flows in the network is varied from 1 to 10, and the same setup as in Figure 6.6 is used. All throughput values are normalized with respect to MCR throughput (the same normalization procedure described before is used). As we can see from the figure, when there is a single flow using CBR traffic, MCR and m-WCETT are comparable[7], because there is no switching in the network. Shortest hop metric throughput is also close to that of MCR because the underlying interface management protocol does a good job of balancing channel assignment. When channel assignments are balanced,the route chosen by the shortest hop metric has a good chance of being channel diverse. In contrast, when there is a single FTP flow, there is a reverse TCP ACK traffic. In this scenario, MCR accounts for the switching cost imposed by forward traffic, and may choose a different reverse route to carry the TCP ACK packets (when forward and reverse traffic use different routes, switching at intermediate nodes is avoided). On the other hand, m-WCETT and shortest hop metric may choose the same route for reverse traffic. Therefore, MCR can perform better than m-WCETT and shortest hop metric.

From Figure 6.10, as the number of CBR flows increase, initially, MCR performs better than m-WCETT. When there are a small number of flows in the network, MCR can find routes that do not require interface switching, and therefore outperforms m-WCETT. As the number of flows becomes extremely large, channels become bottlenecked, and throughput with both the metrics is comparable. On the other hand, with FTP traffic, MCR outperforms w-MCETT, even with a large

[6] We use the default DSR protocol provided for in Qualnet 3.9 to the simulate shortest hop metric, as this version of DSR uses shortest hop routes.

[7] The throughputs may differ for some scenarios because different routes may be chosen by MCR and m-WCETT when multiple routes with the same cost are available.

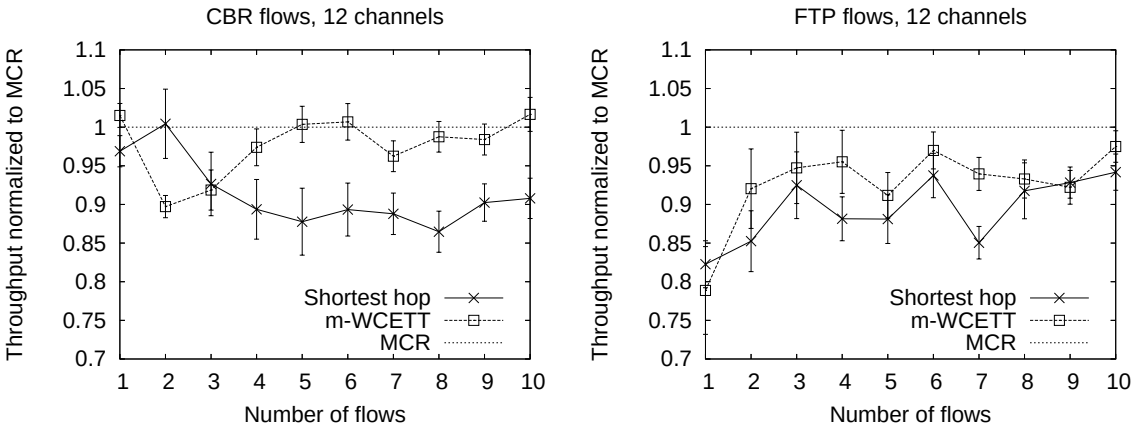

Figure 6.10: Comparison of different routing metrics with both CBR and FTP traffic.

number of flows. MCR reduces end-to-end delay when compared to m-WCETT, by reducing the number of switches, and this translates into better performance with FTP.

Shortest hop metric is consistently worse than MCR as the number of flows increase. Shortest hop metric does not account for either channel diversity or switching costs, and hence does not match the performance of MCR with multiple flows.

In Figure 6.10, each data point was generated by averaging the normalized throughput over 30 different scenarios. Averaging the normalized throughput values allows a simpler comparison between metrics, but hides the variations in normalized throughput for individual scenarios. An alternate way of representing the data is to plot the normalized throughput for each scenario (as a scatter plot), and such a representation is presented in Figure 6.11. As we can see from Figure 6.11, MCR outperforms both m-WCETT and shortest hop for most scenarios. In fact, in certain worst-case scenarios, the throughput with m-WCETT and shortest hop is significantly lower than MCR. There are a few scenarios where m-WCETT and shortest hop outperform MCR, but the magnitude of improvement in those scenarios is small. Therefore, in general, MCR metric is a good choice over a large number of scenarios.

Example scenario to highlight benefits of MCR:

The performance of m-WCETT can be significantly lower than that of MCR in scenarios where multiple routes are available, and some routes require frequent interface switching while others do

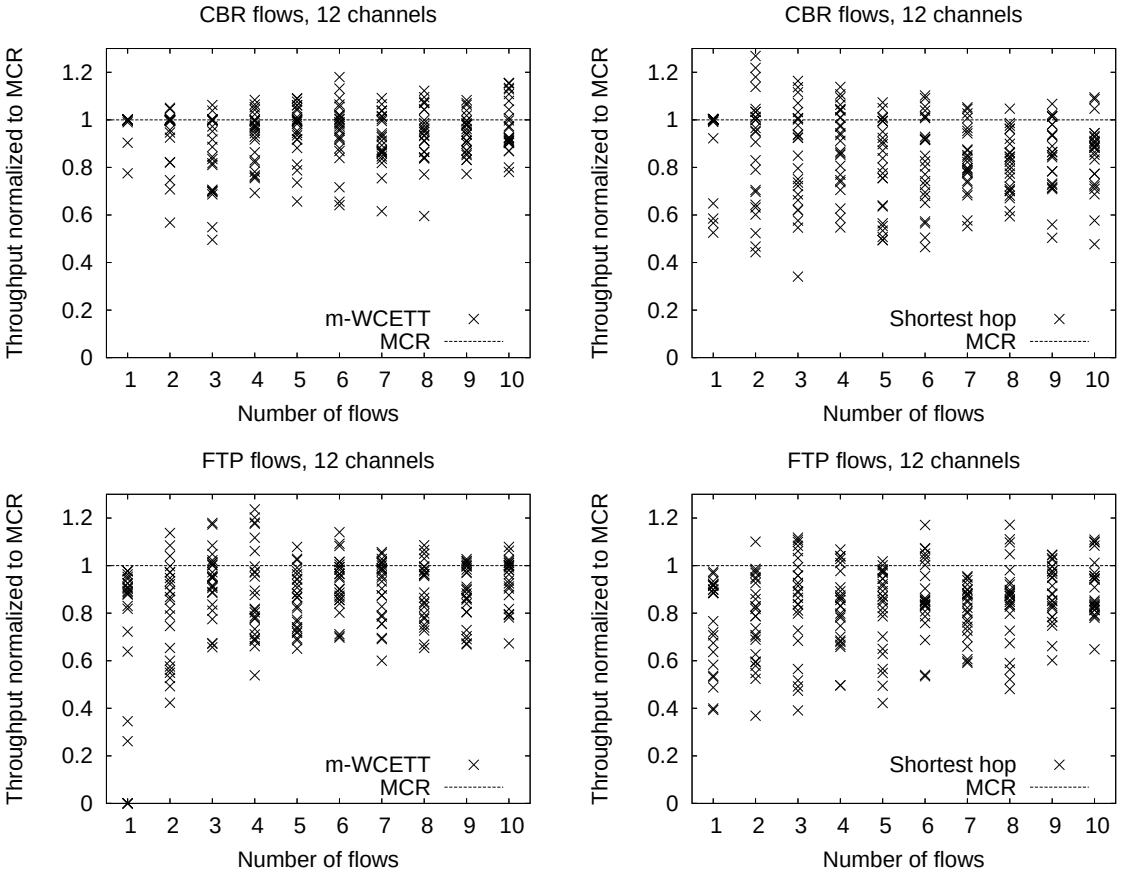

Figure 6.11: A scatter plot comparison of m-WCETT and shortest hop metric with with both CBR and FTP traffic.

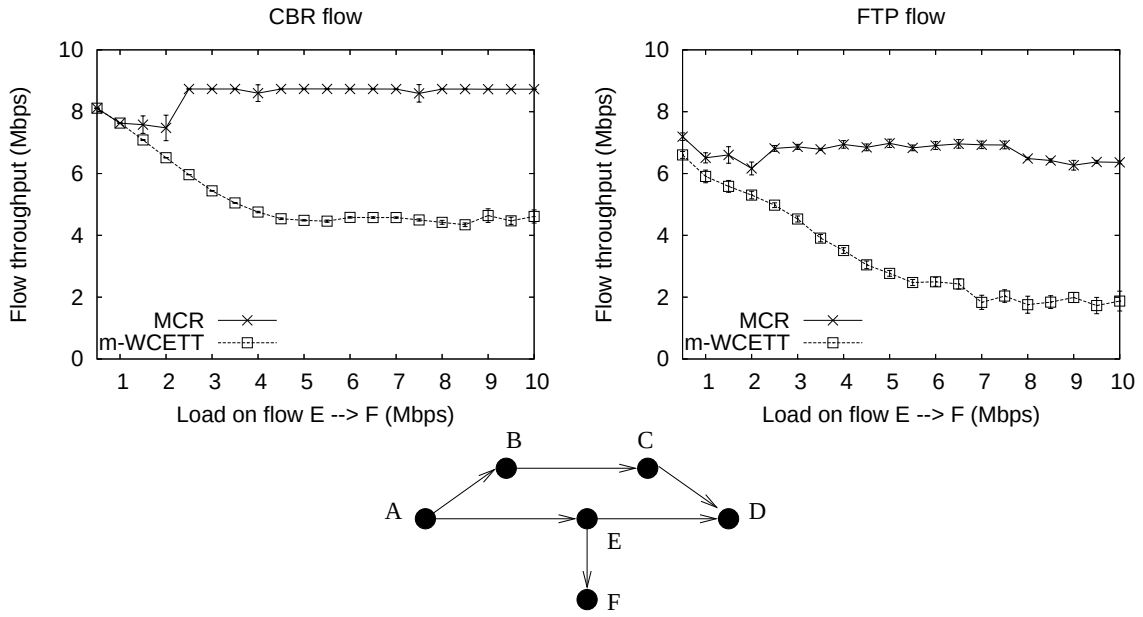

Figure 6.12: Comparison of m-WCETT and MCR with CBR traffic. The load on flow from node E to node F is varied from 0.5 Mbps to 10 Mbps.

not. We highlight this difference by looking at an example scenario shown in Figure 6.12. In the example, a CBR flow is set up from node E to node F. A second flow is set up from node A to node D (the flow is either a CBR flow or a FTP flow), and the nodes are located as shown in Figure 6.12. The figure plots the throughput of the second flow. The experiment is performed for a 2 interface, 12 channel scenario, and the link layer protocol balances channel assignments such that each node is assigned a different channel.

The flow from A to D can use either route A-B-C-D or route A-E-D. Since route A-E-D is shorter (and both hops use different channels), m-WCETT always uses route A-E-D, and the second flow throughput reduces (with increase in load of first flow) because the switchable interface at node E is shared to forward data for both flows. However, the proposed MCR metric uses route A-E-D only when the switching cost at node E is small. When the load on flow E-F is small, the switching cost at node E will be small, and route A-E-D is used. However, when the load increases, switching cost at E exceeds the cost of using an extra hop on the route A-B-C-D. Therefore, MCR then uses route A-B-C-D leading to a constant second flow throughput, independent of the load on the first flow. Furthermore, note that with FTP traffic, even when the first flow load is small, MCR does

better than m-WCETT. This is because, MCR sometimes uses route A-B-C-D for the reverse TCP ACK traffic and route A-E-D for the forward data traffic, while m-WCETT uses the route A-E-D for both forward and reverse traffic.

There are several scenarios, especially with small number of flows in the network, where the use of a longer route to avoid frequent switching is the better choice. Indirectly, by accounting for switching cost, MCR metric balances flows in the network, as recommended by the capacity analysis. In general, MCR metric retains the performance of m-WCETT with large number of flows, while exhibiting superior performance with fewer flows.

6.5 Discussions

In this chapter, we have proposed a new routing metric (MCR) for multichannel wireless networks. The metric is well-suited for the scenario where the number of interfaces per node is smaller than the number of channels. Detailed evaluations have shown that in several scenarios, MCR metric outperforms other routing metrics that have been proposed in the past.

There are several avenues for future work. The MCR metric does not explicitly account for traffic load. This design choice was motivated by the possibility of having frequently changing traffic (such as a large number of short HTTP transfers). Insights from capacity analysis suggest that load balancing may be beneficial for multichannel networks, and in scenarios where the network load is stable and can be estimated accurately, the MCR metric could be extended to be load-aware. Route caching is currently not used in the routing protocol, because the cost of a route can be computed only by using the complete path information. An interesting avenue for future work will be to design alternate multichannel metrics that are decomposable, allowing route information to be cached. The MCR metric is designed as a linear combination of two cost components. Alternate ways of combining the cost components may be another avenue for future work.

CHAPTER 7

IMPLEMENTATION OF MULTICHANNEL PROTOCOLS

Implementing the proposed protocols in realistic testbeds is non-trivial, because popular operating systems do not support many of the features required by the protocols. For example, the use of multiple channels and multiple interfaces, as well as switching interfaces among channels, has to be insulated from existing user applications. At present, there is a lack of architectural support in the kernels to manage multiple channels and interface switching. Therefore, implementing protocols that require frequent switching necessitates additional architectural support in the operating system kernel. In this chapter, we describe the implementation of the proposed multichannel protocols in a Linux-based testbed. First, we survey existing work on building wireless testbeds. Next, we identify the challenges in building a multichannel, multi-interface wireless network, and present our implementation for supporting such a network.

7.1 Related work

There have been several research initiatives on building wireless network testbeds [75–81]. However, there have been relatively fewer attempts at building multichannel wireless testbeds [4, 35, 49, 82]. Of these, [4, 35, 49] assume that interfaces are fixed to a channel for long intervals of time. Therefore, in those implementations, switching interfaces from one channel to another can be done infrequently, possibly using user-space scripts, without requiring support from the kernel. In contrast, our solutions require more fine-grained interface switching, which requires additional architectural support.

"VirtualWifi" [82,83] is a virtualization architecture that abstracts a single wireless interface into multiple virtual interfaces. VirtualWifi provides support for switching the physical interface across

the channels used by each virtual interface. VirtualWifi has some similarity to our implementation, but does not offer all the features necessary for controlled switching among multiple channels. VirtualWifi exports one virtual interface per channel, which *exposes* the available channels (by exposing one IP address per channel) to the user applications, and may necessitate modifying these applications. In contrast, our work *hides* the notion of multiple channels from user applications, and therefore, does not require any modifications to existing applications.

A feature of our implementation is that it exports a single virtual interface to abstract out multiple interfaces. There are other testbed works that can also abstract multiple real interfaces into a single virtual interface [84–86]. However, those approaches are not designed to support the notion of using multiple channels or interface switching between channels.

Architectural changes have been proposed by wireless researchers to support other protocols in ad hoc networks. One well studied architectural problem is to support on-demand routing [75, 76, 86], which requires mechanisms to buffer data packets while a route is being discovered. We implement the on-demand discovery component of the proposed multichannel routing protocol using the same approach as in [76]. However, implementing the other aspects of the multichannel protocols, such as interface management, requires additional support, as they require close interaction with the device drivers.

7.2 Architectural support for interface switching

In this section, we motivate the need for new architectural support in kernel for supporting multichannel protocols that require interface switching. As we noted in Chapter 4, a key goal of this thesis is to develop protocols that can work with existing off-the-shelf hardware. Existing off-the-shelf hardware do allow interfaces to be switched by the driver, but common operating system kernels are not designed to utilize this feature. Operating systems have always tried to abstract out the details of the underlying hardware from higher layer applications. We want to continue to preserve this design principle, which in turn requires abstracting out the details of multiple channels and interface switching from user applications and higher layers of the network stack. These requirements necessitate changes to the operating system kernel. As we argue next, supporting

interface switching in multichannel networks requires non-trivial changes to the kernel. Our approach is to develop a generic *channel abstraction layer* to support interface switching. The channel abstraction layer could be used to implement other multichannel protocols that require interface switching, in addition to the protocols proposed in this thesis.

7.2.1 Need for new support

We identify the features needed to implement interface switched multichannel protocols (that are missing in current operating systems) by using Linux as an example. The key features that are lacking in Linux are as follows:

1. Specifying the channel to use for reaching a neighbor:

Common operating systems do not allow the network layer to control the channels (or other radio features, such as data rates, transmission powers, and antennas) used to reach a neighboring node. Instead, the kernel routing tables only provide control over the interface to use to reach a neighboring node. In a single channel network, there is no benefit in explicitly selecting channels because all nodes have to use a common channel. This lack of explicit channel support is not a problem in those multichannel networks where each interface is associated with exactly one channel, i.e., there is an one-to-one mapping between interfaces and channels. For example, there is an one-to-one mapping between interfaces and channels in a network where each node has m interfaces, and the interfaces of a node are always fixed on some m channels. In this setting, the channel to use to reach a neighboring node can be indirectly specified by the specifying interface to use, since each interface is associated with a unique channel.

However, in the scenario we address in this thesis, the number of interfaces per node could be significantly smaller than the number of channels. Therefore, there is no longer an one-to-one mapping between channels and interfaces. Under this scenario, we have shown that a good strategy is to use the same (switchable) interface to send data to different neighboring nodes over possibly different channels. To support the notion of switchable interfaces, we need to to control the channels to use to reach a node.

For example, consider the scenario shown in Figure 7.1. In the figure, suppose that each node

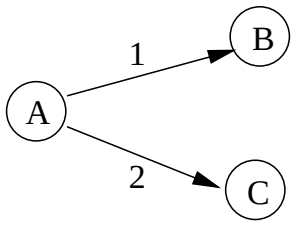

Figure 7.1: Example illustrating the lack of kernel support for multichannel protocols.

has a single interface. Also suppose that node B is listening to channel 1 and node C is listening to channel 2. Under this scenario, when A has to send some data to B, it has to send the data over channel 1, and similarly data to C has to be sent over channel 2 (the interface at A has to be switched between channels 1 and 2, when it is so required). This example shows that the channel to use for transmitting a packet may vary based on destinations, even if the same interface is used for all destinations. In general, without support for specifying the channels to use to reach a neighboring node, it is difficult to implement those multichannel protocols that use a *single* interface to send data to different neighboring nodes over *different* channels.

2. Specifying channels to use for broadcast:

In a single channel network, broadcast packets sent out on the wireless channel are typically received by nodes within the transmission range of the sender. The wireless broadcast property is used to efficiently exchange information with multiple neighbors (for example, during route discovery). In a multichannel network, different nodes may be listening to different channels. Therefore, to allow broadcast packets in a multichannel network to reach all the nodes that would have received the packet in a single-channel network, copies of the broadcast packet may have to be sent out on multiple channels. For example, in Figure 7.1, node A will have to send a copy of any broadcast packet on both channel 1 and channel 2 to ensure that its neighbors B and C can receive the packet.

There are several existing applications that use broadcast communication, for example, the address resolution protocol (ARP). To ensure that the use of multiple channels is transparent to such applications, it is necessary that the kernel send out copies of broadcast packets on multiple channels, when necessary. However, there is no support in the existing kernel to specify which

channels broadcast packets have to be sent out on, or to actually create and send out copies of broadcast packets on multiple channels. Therefore, there is a need to incorporate mechanisms in the kernel for supporting multichannel broadcast.

3. Buffering and scheduling support

As we discussed earlier, interfaces may have to be switched between different channels to enable communication among neighboring nodes that are on different channels, and to support broadcasts. A switch is required when a packet has to be sent out on some channel c, and at that time there is no interface tuned to channel c. Suppose that the kernel can decide whether a switch is necessary to send out some packet. Even then, the kernel has to decide whether an immediate switch is feasible. For example, if an interface is still transmitting an earlier packet, or has buffered some other packets for transmission, then an immediate switch may result in the loss of those packets that are awaiting transmission in the interface queue. Therefore, there is a need for mechanisms in the kernel to decide if earlier transmissions are complete, before switching an interface.

When an interface cannot be immediately switched to a new channel, packets have to be buffered in a channel queue until the interface can be switched. Switching an interface incurs a non-negligible delay (around 5 ms with the interfaces used in our testbed), and switching too frequently may significantly degrade performance. Therefore, there is a need for a queuing algorithm to buffer packets, as well as a scheduling algorithm to transmit buffered packets using a policy that reduces frequent switching, yet ensures queuing delay is not too large.

7.2.2 Design choices

The earlier discussions clearly identify the need for several new features in the kernel for supporting the use of multiple channels, especially when interfaces have to switch between channels. The Linux kernel's networking stack is organized into multiple layers to ease implementation and improve extensibility. For example, IP belongs to the network layer, while the device drivers that control access to the interface hardware are part of the link layer. Once we have decided to add support for interface switching, the next question is to identify the layer where interface switching support can be added. Interface switching support requires close interaction with the interface device driver. Based on this requirement, we have three possible locations for adding interface switching support:

1. Add interface switching support directly into the device driver. This approach offers the most control in accessing the interfaces, but has two main drawbacks. First, this approach ties in our implementation with a specific device driver. Second, multiple interfaces cannot be cleanly handled within the device driver of a single interface.

2. Add interface switching support into the network layer (for example, as a "Netfilter" hook [87]). This approach insulates the implementation from the specifics of device drivers. However, multiple interfaces are visible to the network layer, and this may require modifications to some protocols that are at (or below) the network layer (such as ARP).

3. Add interface switching support as a new module that operates between the network layer (as well as ARP) and the device drivers. The module may be logically viewed as belonging to the link layer. This approach has the benefit of being insulated from device driver specifics, while presenting a single virtual interface to the network layer. The virtual interface can abstract multiple interfaces that may be actually available, and insulates the network layer from the need to know the details of the number and types of interfaces. We choose this approach, and implement a new *channel abstraction layer* module.

The option we have chosen has some additional benefits. Linux already has the ability to "bond" multiple interfaces into a single virtual interface using a link layer "bonding driver" that resides between the network layer and the device drivers. The bonding driver is typically used for grouping multiple Ethernet-based devices into a single virtual device. The bonding driver offers features that allow for load balancing (striping) over the available interfaces, interface fail-over support, etc. There is also a set of user space tools which support management operations, such as specifying which real interfaces to group into a single virtual interface. We have implemented the channel abstraction layer as a new feature of the bonding driver. In the next section, we describe the implementation of the multichannel protocols proposed in this thesis using the interface switching support provided by the channel abstraction layer.

7.3 Implementation

The multichannel protocols proposed in this thesis have been implemented on a Linux-based testbed. In this section, we will first describe the implementation architecture, and then describe the implementation of each of the key components.

7.3.1 Implementation architecture

The multichannel implementation architecture is shown in Figure 7.2. Our implementation has three main components, which collectively implement the proposed interface management and routing protocols.

- Channel abstraction layer: This kernel component provides support for fast interface switching. This component is generic enough to support other multichannel protocols. The channel abstraction layer abstracts the details of multiple channels and interfaces from the higher layers, and is controlled by "IOCTL" commands from the userspace daemon.
- Kernel multichannel routing support: This component is used to provide kernel support for on-demand routing. The component informs the userspace daemon when a route discovery has to be initiated, and buffers data packets while the route discovery is pending.
- Userspace daemon: The userspace daemon implements the less time-critical components of the interface management and routing protocols. Most of the protocol functionality is implemented in this component.

The kernel components interact with the Linux TCP/IP implementation and the interface device drivers, while the userspace daemon is built using standard userspace networking libraries. Most of the multichannel protocol has been built into the userspace daemon. The kernel components support only a small set of essential features. In the rest of this section, we describe the implementation of each component, as well as the interaction between components.

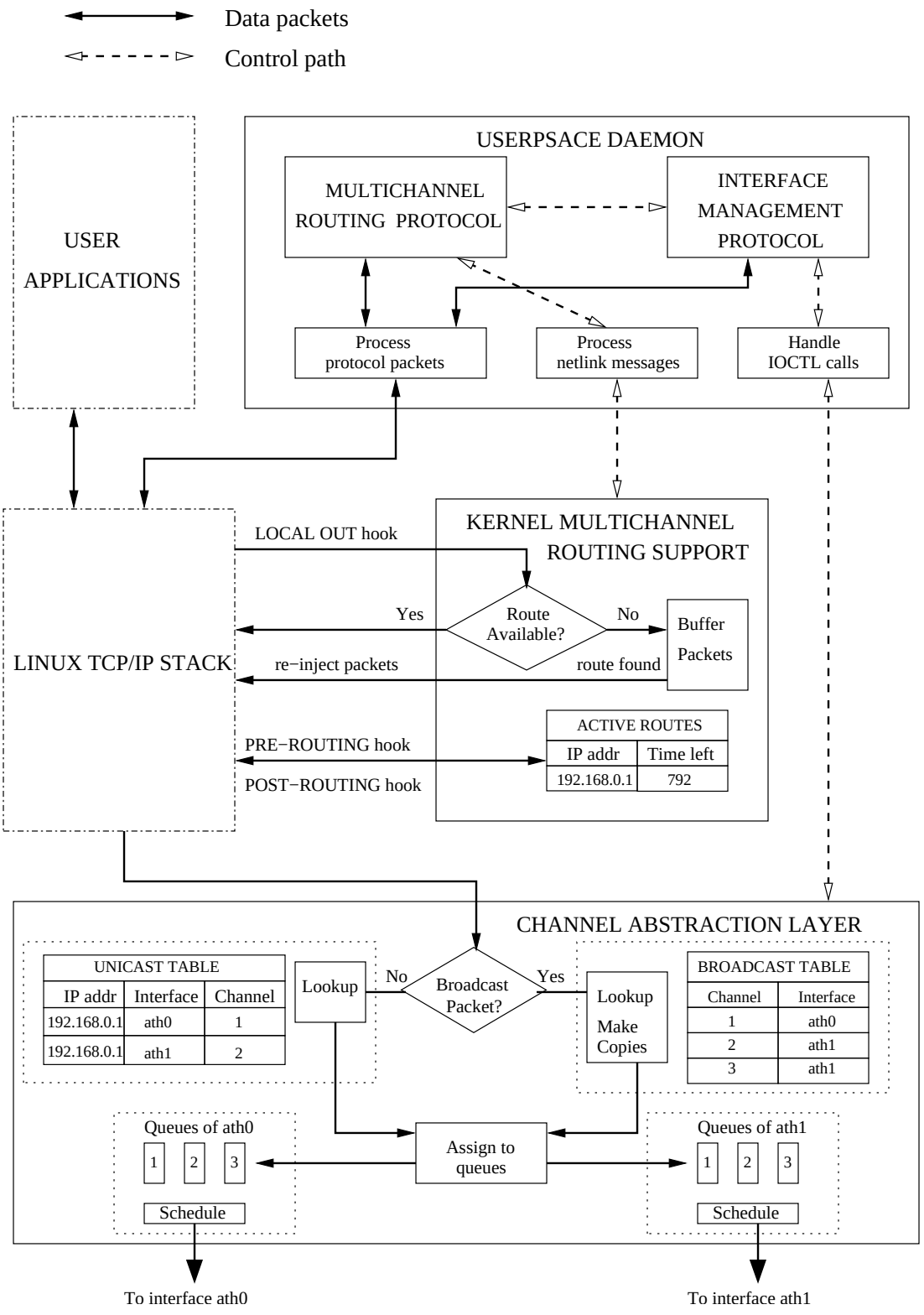

Figure 7.2: Architecture for implementing multichannel protocols. The figure assumes that two interfaces, “ath0” and “ath1”, are available.

IOCTL call	Function
AddValidChannel	Specify the channels that may be used by an interface.
UnicastEntry	Add, update, or modify an entry in the unicast table.
BroadcastEntry	Add or remove an entry in the broadcast table.
SwitchChannel	Explicitly switch an interface to a new channel.
GetStatistics	Return per-channel usage statistics.

Table 7.1: List of IOCTL calls exposed by channel abstraction layer.

7.3.2 Channel abstraction layer

The channel abstraction layer (CAL) was implemented by a member of our research group, Chandrakanth Chereddi, and the details of the implementation are in his thesis [12]. Here, we outline the design of CAL, and the mechanisms for using the layer.

CAL maintains a "unicast table" (see Figure 7.2). The unicast table contains the channel and interface to use for sending data to any destination. There is a second table, called the "broadcast table", which contains a list of channels and interfaces on which copies of broadcast packets have to be sent out. Entries in both the tables are set by the userspace daemon (details below). Each interface is associated with a set of channel queues. When CAL receives a packet from the network layer, if the packet is a unicast packet, then the packet is inserted into the appropriate channel queue after consulting the unicast table. Otherwise, if the packet is a broadcast packet, copies of the packet are inserted into all channels listed in the broadcast table. CAL implements a scheduler to decide when interfaces switch between channels. The scheduler implements the scheduling strategy described in Section 5.3.1. CAL also requires a few modifications to the interface device drivers to optimize performance. More details are in [12].

The userspace daemon communicates with CAL using a set of IOCTL calls. IOCTL calls are a common way in Linux for interaction between kernel and userspace components. The list of IOCTL calls provided by CAL and their functionality is listed in Table 7.1. We will describe later the sequence in which these IOCTL calls are invoked by the userspace daemon.

7.3.3 Kernel multichannel routing support

The kernel multichannel routing (KMCR) module provides support for on-demand routing. For example, when an application initiates communication to a destination that is not a direct neighbor,

a new route may have to be setup if no route to the destination is already available. The route discovery protocol is implemented as part of the userspace daemon. However, a mechanism is needed to invoke the discovery process when a new route is desired by the application. Clearly, the only place where access to all application packets is available is in the kernel. Therefore, several earlier routing implementations [75, 76, 86] have included support in the kernel to initiate route discovery.

The on-demand route discovery process should be transparent to applications. While the on-demand route discovery is in progress, any packets sent by the application have to be buffered, and later sent out once a new route is available. This buffering is required to prevent packet drops. Note that higher-layer protocols such as TCP are severely affected by the loss of the initial packets in a connection. For example, TCP incurs a large timeout if the initial SYN packet, used for connection establishment, is lost. Packet buffering has been implemented in past works in two different ways. In one approach [86], application packets are sent up to the userspace and stored by a userspace daemon, and re-injected once the route is discovered. In the second approach [76], packets are buffered in the kernel itself. We follow the second approach because it avoids the context switching overheads of sending a packet up to the userspace.

The kernel routing support is implemented as a module which can be loaded into the Linux kernel. The module utilizes the Linux Netfilter support [87], and the implementation was based on the AODV implementation from Uppsala university [76, 88]. Figure 7.2 shows the structure of KMCR module. The KMCR module maintains a "active route table" containing a list of nodes to which routes are available. Each node in the list is also associated with a "time left" field that represents the time period after which the route to that node is deemed to be inactive.

The Netfilter library offers "hooks" to intercept packets traversing through the networking stack. The "LOCAL OUT" hook intercepts packets that originate in the node, before the routes to be used by the packets are computed. The KMCR module adds itself to the LOCAL OUT hook. If a packet received on the LOCAL OUT hook is destined for a node (using the wireless interface) that does not currently have a route, then a new route has to be discovered. Otherwise, if a route exists, the packet is returned without any modifications. When a route is not available, KMCR

requests the userspace daemon to find a new route. After that, any packets to that destination are captured from the LOCAL OUT hook and buffered in the KMCR module until a new route is discovered. After a route is discovered, the KMCR module is notified by the userspace daemon, which then re-injects the buffered packets. If the route discovery fails, then any packets that had been buffered, pending route discovery, are dropped.

The KMCR module also adds itself to two other netfilter hooks; the "PRE-ROUTING" hook and the "POST-ROUTING" hook. The PRE-ROUTING hook intercepts packets received by a node from an external node, while the POST-ROUTING hook intercepts packets that are destined for an external node. Packets received on the PRE-ROUTING hook over the switchable interface are dropped (as the switchable interface is not intended for receiving data). Suppose X is a node corresponding to either the source of a packet intercepted on the PRE-ROUTING hook (that is received over the fixed interface), or the destination of a packet intercepted on the POST-ROUTING hook. Then, the time left for the route to X, contained in the active route table, is reset to a maximum *LifeTime* value (we set the *LifeTime* value to 30 seconds in our implementation). Essentially, when packets are intercepted on the PRE-ROUTING and POST-ROUTING hooks, it indicates that the routes used by the packets are active. If no packet is intercepted along a route over a time longer than the *LifeTime* value, then the route is assumed to be not in use. The userspace daemon periodically checks which routes in the active route table are no longer in use, and removes them.

The communication between the KMCR module and the userspace daemon is implemented using "netlink" messages. Netlink library is a feature in Linux to support communication between kernel and userspace. Netlink offers more flexibility than IOCTL calls by allowing two way communication, and is especially useful when more than a few bytes of information have to be exchanged. The list of netlink messages implemented by the KMCR module, and their functionality is listed in Table 7.2.

7.3.4 Userspace daemon

The userspace daemon implements the interface management and routing protocols by utilizing the features offered by the CAL module and the KMCR module. In this section, we omit the details

Netlink message	Function
AddRoute	Add an entry into active table. Implies route discovery was successful.
DiscoveryFailed	Inform KMCR that route discovery failed.
DeleteRoute	Remove an entry from active route table.
InitiateDiscovery	Request sent by KMCR to userspace for initiating route discovery.
IsRouteActive	Query KMCR if the requested route is active.
RouteStatus	Response from KMCR to userspace on the status of the requested route.

Table 7.2: List of netlink messages supported by the kernel multichannel support module.

of the protocols themselves as they have been presented earlier in the thesis, and instead focus on issues specific to the testbed implementation. The implementation currently supports at most two interfaces per node, but can be easily extended to support more than two interfaces. The userspace daemon should be started only after the CAL and KMCR modules have already been loaded into the kernel. Our current userspace implementation assumes that two interfaces are available at each node (and can be easily extended to handle more than two interfaces). In the next section, we describe extensions to handle nodes with a single interface.

Initialization: The userspace daemon is provided with a list of valid channels and a list of available interfaces in a configuration file. Using this information, the daemon initializes the kernel modules as shown in Figure 7.3. The initialization protocol first informs CAL of the set of valid channels associated with each interface. Next, using the fixed channel selection protocol (Section 5.3.2) one of the available channels is randomly chosen as the fixed channel, and one of the interfaces is switched to the fixed channel, while the second interface is switched to any of the remaining channels. After that, the broadcast list is initialized such that the fixed interface is used to transmit on the fixed channel, and the switchable interface is used to transmit on the remaining channels. Once these initialization steps are complete, the hello packets announcing the fixed channel can be sent out. After that, whenever the fixed channel of the node is changed, a request is sent to the CAL to switch the fixed interface.

Managing received hello packets: As described in Section 5.3.2, hello packets received from a neighbor enables a node to discover the channels used by its neighbor. When the fixed channel being used by a neighbor is first discovered, a new entry is added into the unicast table of CAL by sending a UnicastEntry message. Later, if the channel used by the neighbor changes, CAL

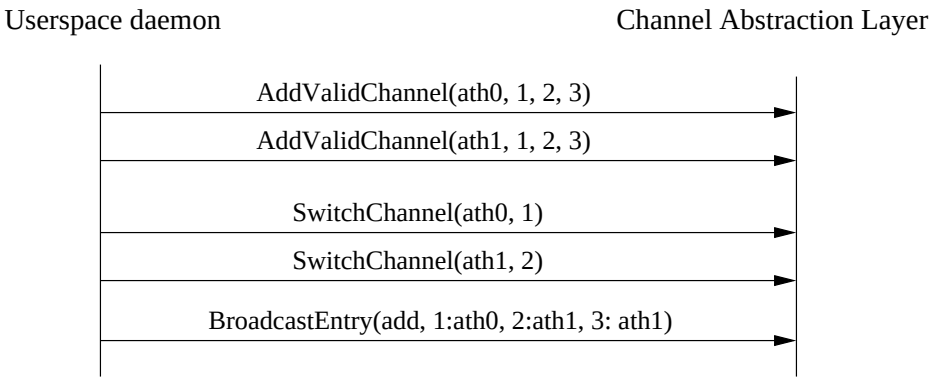

Figure 7.3: Commands invoked at initialization. The figure assumes that two interfaces (ath0, ath1) and three channels (1, 2, 3) are available. The values in parenthesis are parameters included in the IOCTL calls. For example, the values in the BroadcastEntry call list the interface to use for each broadcast channel.

is updated by another UnicastEntry message. Similarly, if no hello messages have been received from a neighbor for more than a timeout duration, then the entry corresponding to the neighbor is removed from the CAL using a UnicastEntry message.

Route discovery and maintenance: The route discovery process is initiated by the KMCR module, as we described earlier. Figure 7.4 shows the interaction between the KMCR module, the userspace daemon, and the CAL module to set up a new route. When the KMCR module at some node S discovers the need for a new route to some node D, it sends an InitiateDiscovery request to the userspace module. The userspace module then queries the CAL module to obtain the switching cost of using different channels (recall that switching cost of channels is used by the routing metric). After that, a route request (RREQ) packet is sent out. Intermediate nodes on receiving the RREQ again query their CAL to obtain the switching cost, which is included while forwarding the RREQ. The destination on receiving the RREQ responds with a route reply (RREP) packet. The source node S on receiving the RREP adds a new entry in the unicast table of the CAL for node D, and the channel to use for node D is set to the channel used to reach the first hop node on the route to node D. After that, the KMCR module is informed of the successful route discovery through an AddRoute message. If a route discovery fails, then the KMCR module is informed of the failure using the DiscoveryFailed message.

Intermediate nodes that forward the RREP containing the route from S to D have to also add

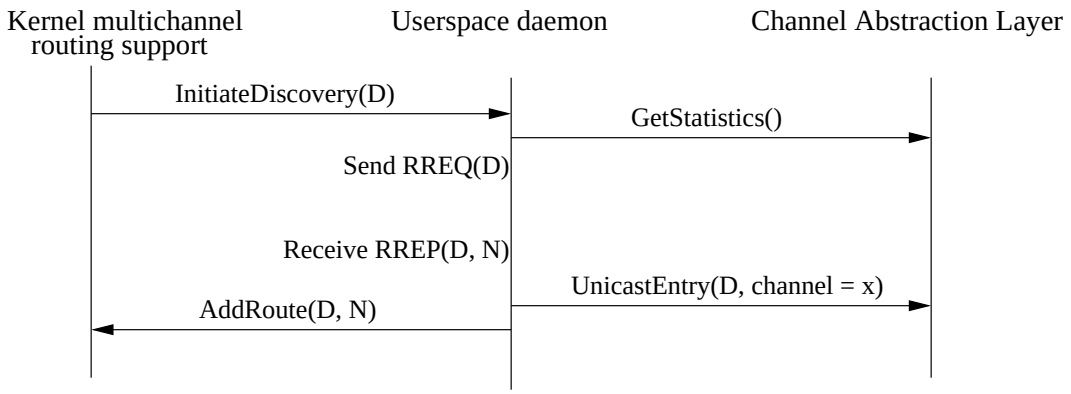

Figure 7.4: Commands invoked during route discovery. The example assumes that a route is being discovered to node D, and the next hop on the discovered route is node N.

information about the route into their kernel tables. Typically, after a route is set up from S to D, the route is used for bi-directional communication. Therefore, intermediate nodes have to add a route to both the source S and the destination D to ensure that data packets between S and D, sent in either direction, are forwarded. The process of adding a route is similar to the procedure followed by node S, which was described above (though the process has to be invoked twice to add routes to both S and D).

Route maintenance involves removing routes from the KMCR table that have not been active for a long time or have been identified as broken. Inactive routes are discovered by the userspace daemon by periodically querying the KMCR module with the IsRouteActive message (the response is received in the RouteStatus message). Broken routes are identified when the next hop is not reachable, or when a route error message is received. In all cases, the route is removed from the active route table of KMCR by sending the DeleteRoute message, and from the unicast table of CAL by sending the UnicastEntry message.

The multichannel protocol implementation described above provides most of the support required for running a multihop multichannel network, provided each node is equipped with two interfaces. The protocols allow any node in the network to communicate with any other node. In our work, we have assumed that there is a separate address assignment protocol to assign addresses to individual nodes. Any of the address assignment protocols proposed in the literature [89], or even manual assignment (if appropriate), could be used with our implementation. Our testbed

experiments have used manual address assignment, and we defer to future work automated address assignment.

7.4 Mesh networking extensions

Multihop wireless mesh is one network architecture for providing last mile wireless connectivity. This architecture has been used in building community wireless networks [90,91] city-wide wireless networks [92], among others. In this section, we describe extensions to our implementation to support mesh networking. The mesh networking implementation demonstrates the feasibility of using multiple channels in a mesh network.

Figure 7.5 describes the mesh networking architecture that we support. Nodes in the network are classified into two types; "mesh nodes" that run the software we have implemented, and "client nodes" that are unmodified. Client nodes are equipped with one IEEE 802.11 interface, while mesh nodes have either one or two interfaces. Some of the mesh nodes are connected to the Internet and act as "gateways". Mesh nodes form a backbone network that is fully connected, and a mesh node is capable of communicating with any other mesh node. A client node connects to one of the mesh nodes that is in its direct communication range. The mesh nodes are viewed by the clients as access points. All mesh nodes and client nodes are capable of communicating with hosts in the Internet, but the client nodes do not communicate with any mesh node other than the node they are connected to directly.

Mesh nodes equipped with two interfaces support the multichannel protocols presented in this thesis, and can potentially transmit on any channel. Single interface mesh nodes support a modified interface management protocol which requires them to be fixed on a specific channel, but support the multichannel routing protocol described earlier. In the rest of this section, we describe extensions to the implementation to provide support for gateways, support for mesh nodes that have a single interface, and support for allowing unmodified client nodes to connect to the Internet through a mesh node.

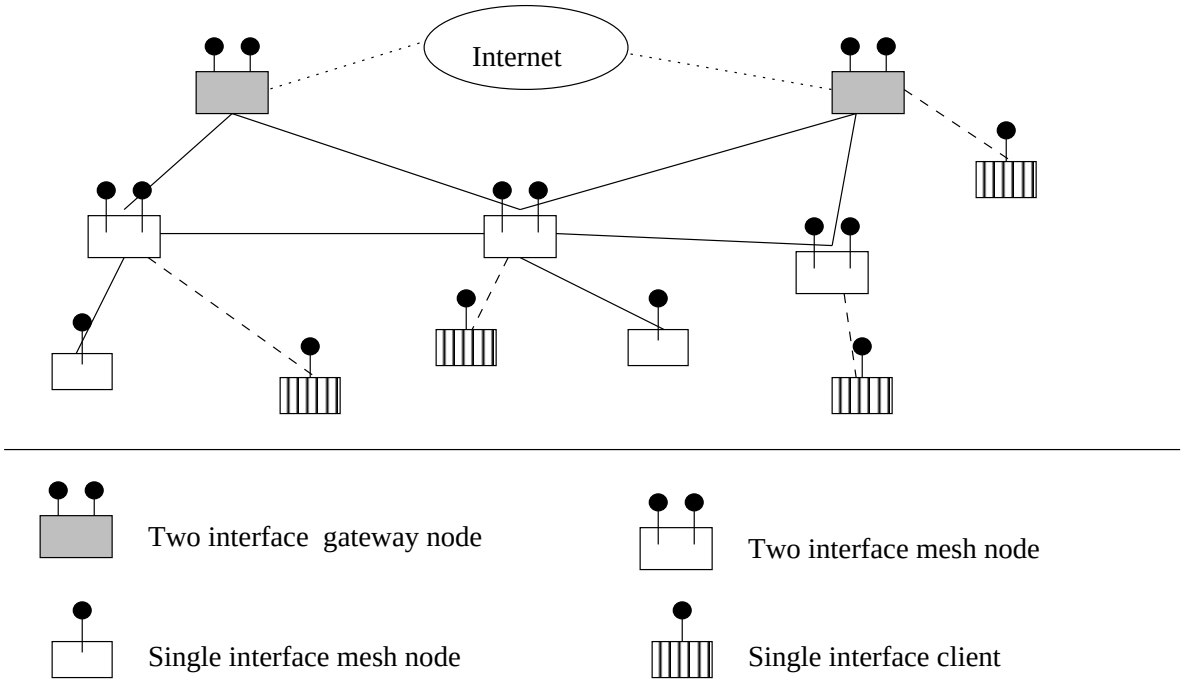

Figure 7.5: The mesh networking architecture supported by our implementation.

7.4.1 Gateway support

A gateway node uses a wired interface to connect to the Internet, while using two wireless interfaces to participate in the mesh backbone. In our testbed, the mesh network uses a private address space that is not visible to the Internet. Therefore, to allow nodes in the mesh network to communicate with nodes in the Internet, network address translation (NAT) is required at the gateway node[1]. Standard Linux distributions provide NAT support, and the translation can be set up using the "iptables" tool.

When an application on a mesh node initiates communication with a node in the Internet, the packets have to be first routed from that node to the gateway node, and from there on to the Internet. IP packets have a destination field in the IP header containing the address of the destination, and this is used by the routing tables in the mesh nodes for deciding the channel and next hop node to use while forwarding a packet. When a packet is destined to the Internet, the gateway is an intermediate destination, with the actual destination located in the Internet. If the

[1]The address translation ensures that packets originating in any mesh node appear to the nodes in the Internet to be originating from the gateway node, on the wired interface

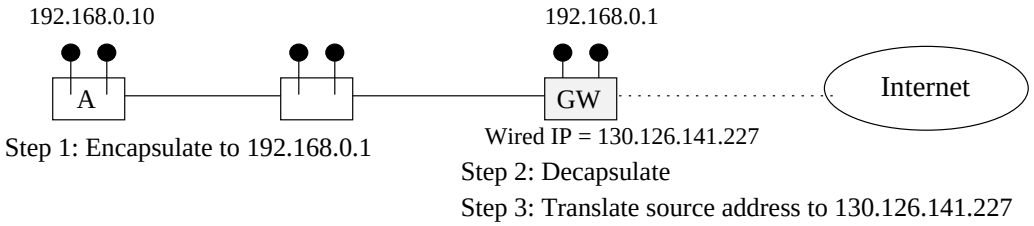

Figure 7.6: Steps involved in sending a packet to the Internet. Source node A encapsulates the packet, which is forwarded to the gateway node GW. Gateway node decapsulates the packet, translates the address and forwards the packet to Internet.

destination address in a IP packet is set to that of the final destination, then there is no information in the packet to guide it toward the gateway node. On the other hand, if the destination address is set to that of the gateway node, then the final destination address is not contained in the packet.

To ensure that a packet contains the address of both the gateway node and the final destination (without requiring modifications to IP), we use a *packet encapsulation* approach (a similar approach was used by [88]). At the source mesh node, in the first step, an IP packet is created with the destination address in the IP header set to the address of the node in the Internet. After that, the packet is encapsulated with a second IP header[2], and the destination address in the second header is set to that of the gateway. When the packet reaches the gateway, the gateway node decapsulates any packets that are encapsulated, and then forwards the packet on to the Internet. The process of encapsulation/decapsulation and network address translation is shown for an example scenario in Figure 7.6.

The support for encapsulation and decapsulation is implemented in the KMCR module. The encapsulation is done in the LOCAL OUT netfilter hook at each mesh node (only for packets destined to the Internet), while the decapsulation is done in the PRE-ROUTING netfilter hook (only at gateway nodes). Note that packets that are coming in from the Internet to a mesh node do not need encapsulation, because the network address translation implicitly ensures that packets to mesh nodes are first routed to the gateway node.

Discovering gateway nodes: A gateway node sends out a broadcast advertisement informing

[2]This mechanism is provided for in the IP protocol, and is called the IP in IP mode. The mode is often used to tunnel packets over multiple IP hops, for supporting mechanisms such as mobile IP.

other nodes that it is a gateway. The advertisement message has a "hop" field containing the number of hops to the gateway node. Initially, the field is set to zero, and every node that forwards the message increases the hop field by one. The message is forwarded only up to a pre-specified maximum number of hops which can be suitably chosen. A node receiving the advertisement message will learn the address of the gateway node and the number of hops to the gateway. Similar gateway discovery mechanisms have been proposed in the past as well [89].

Our implementation supports multiple gateways in the mesh network. Having multiple gateways provides resilience to failures, and allows traffic to the Internet to be distributed across nodes. When a node receives advertisement messages from multiple gateways, it selects one of the gateways as the gateway it intends to use. In our implementation, the gateway that is fewest hops away is selected, with ties broken arbitrarily. The gateway selection mechanism can be easily extended to use other cost metrics, instead of basing the selection on the number of hops to the gateway.

7.4.2 Single interface support

Mesh nodes that are equipped with a single interface cannot support the hybrid interface assignment strategy which requires at least one fixed interface and one switchable interface. It is possible that a mesh network may be incrementally deployed, with some of the nodes having only a single interface. We aim to allow single interface nodes to participate in the mesh network, though at the cost of not being able to communicate directly with all neighbors.

A node S with a single interface keeps its interface fixed on one of the available channels (the algorithm for selecting the fixed channel is described below). The fixed channel is chosen such that the node S has at least one multi-interface node, say M, that can be directly reached on the fixed channel. Single interface nodes implement the full multichannel routing protocol presented earlier. Therefore, if the single interface node has to communicate with any other neighboring node that is not sharing a fixed channel with itself, it can do so by routing via node M. This ensures that node S can communicate with any other node in the network via node M.

During initialization, a single interface node S first randomly selects a channel and switches to that channel. After that, node S sends out a broadcast message on that channel advertising its

presence. Any multi-interface mesh node M that receives this advertised message[3] responds back with a unicast acknowledgment announcing its presence. If node S receives an acknowledgment, node S learns the availability of a multi-interface node M on that channel. Node S then fixes its single interface to the channel it is already on, completing the fixed channel selection. On the other hand, if node S does not receive any acknowledgments, it learns that there are no multi-interface nodes on its current channel within its communication range. Node S then switches its interface to the next available channel and repeats the discovery procedure, and the process continues through the available channels till a channel with a multi-interface neighbor is found. After initialization, the fixed channel selection algorithm is repeated whenever no multi-interface nodes are reachable on the fixed channel. The discovery procedure could be extended to select fixed channels based on other metrics, such as the number of multi-interface nodes on a channel, or the load on a channel.

The fixed channel selection procedure described above allows a single interface node to be connected to the mesh network if it has at least one multi-interface node in its neighborhood. This requirement is conservative as it does not allow for a single interface node S to connect to the rest of mesh network through another single interface node, say X. Note that if X is connected to mesh backbone, then S could connect to the mesh backbone through X. We do not allow a single interface node to be connected through another single interface node because of the possibility of network partitions. For example, node X may believe it is connected to the mesh backbone through node S, while node S believes it is connected through node X. In that scenario, nodes S and X could be connected to each other, but not to other nodes in the mesh network. Then, even though the mesh nodes form a connected network when all nodes use a common channel, they may no longer be connected if some single interface nodes are only connected to other single interface nodes. It is possible to have a more elaborate protocol which ensures that a set of connected single interface nodes are also connected to at least one multi-interface node, but we defer such an extension to future work.

[3]Note that a multi-interface node M would receive the message only if the channel on which the message has been sent happens to be M's fixed channel.

7.4.3 Support for client nodes

Client nodes do not run any of the software that we have implemented. Instead, client nodes view mesh nodes as "access points" and connect to them using standard IEEE 802.11 protocol rules. A client node is allowed to connect to any of its neighboring mesh nodes on the fixed interface of the mesh node. In typical wireless networks, clients connect to access points using the "managed" mode provided for in IEEE 802.11. However, in our implementation mesh nodes are connected to each other using the "ad hoc" mode, and therefore a client will have to connect to the fixed interface of a mesh node in the "ad hoc" node. An alternate solution would be to have an extra interface at each mesh node, and set up the extra interface to operate as an access point in the managed mode. Then, clients could connect to the extra interface in the managed mode. Our approach (of connecting to the fixed interface) avoids the need for an extra interface.

Our implementation requires the mesh nodes to be assigned addresses with a common subnet prefix, while client addresses must belong to a different subnet prefix. The userspace daemon adds a default entry into the unicast table of the channel abstraction layer. The default entry ensures that packets to any destinations not in the unicast table are sent out on the node's fixed channel. This ensures that packets sent to clients are sent out on the fixed interface (because no entries are added into unicast table for clients). Clients have to be configured to connect to any of the mesh nodes in its vicinity on the fixed channel of the mesh node[4]. In our implementation, clients are allowed to only communicate with nodes in the Internet, or with other nodes in the mesh network. However, our implementation could be extended to allow mesh nodes to connect to client nodes as well. Packets sent out by a client are network address translated by the mesh node to which it is connected, and then sent on to the Internet or another mesh node. Figure 7.7 illustrates the process of a client communicating with a node in the Internet. Apart from the minimal configuration requirements, clients can run without requiring any other changes.

[4]The standard IEEE 802.11 beacons continue to be sent out on the fixed channel in our implementation, and enables clients to discover the channel on which a mesh node is available.

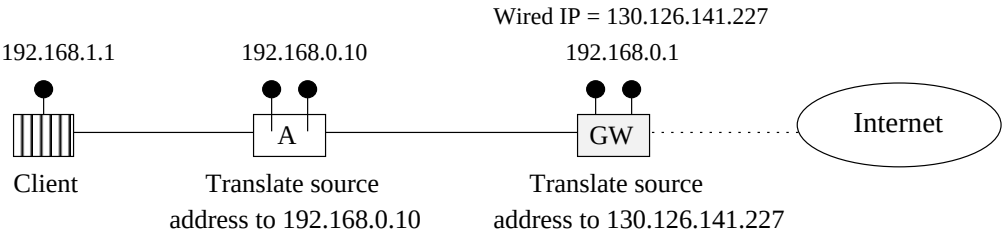

Figure 7.7: Client communication process. Client packets are network address translated at its mesh router, node A. Example assumes that client address space is 192.168.1.*, while mesh node address space is 192.168.0.*.

Figure 7.8: Picture of a testbed node. Each testbed node is equipped with two wireless interfaces.

7.5 Testbed experimentation

The multichannel implementation has been evaluated on a testbed that we have built. Nodes in the testbed comprise of *Net 4521* boxes from Soekris [10]. Figure 7.8 shows a picture of a testbed node. Each node is currently equipped with two wireless interfaces (one pcmcia interface and one mini-pci interface), and it is possible to have up to 3 wireless radios using this hardware platform. The wireless cards are based on *Atheros* chipset [11], and support IEEE 802.11a/b/g protocols.

The multichannel implementation has been carefully tested for correctness on the testbed. We have set up scenarios with gateway nodes, single interface mesh nodes and client nodes, and verified the correct operation of the protocols. The performance of channel abstraction layer has been extensively studied in Chereddi's thesis [12]. The CAL performance was measured while operating it as part of the multichannel implementation described in this thesis. Therefore, evaluation results from [12] also measure the performance of our implementation (we do not restate those results here),

and the results there demonstrate significant performance improvements when multiple channels are used. In the rest of this section, we present sample results to demonstrate the benefits of our implementation in single hop and multihop scenarios.

7.5.1 Single hop experiments

We measure the performance of the multichannel implementation under a single hop scenario. Five nodes are placed such that each node can directly communicate with any other node. The nodes are numbered from 0 to 4 and node i sets up a flow to node $(i + 1)$ mod 5. We vary the number of channels in the network from one to four. All nodes are equipped with two interfaces. The channel data rate is set to 6 Mbps. The experiments are run with both TCP and UDP traffic (flows are run for 100 s), and traffic is created using the iperf tool [93]. Iperf creates back-logged traffic.

Figure 7.9 plots the aggregate throughput for both TCP and UDP traffic. As we can see from the figure, when more channels are available, there is a substantial improvement in the throughput. With UDP traffic, each node has data to transmit on exactly one channel. Therefore, interface switching is not required at any of the nodes. However, with TCP traffic, each node has to send TCP data packets along the flow it is initiating, and send TCP ACK packets along the flow it is receiving data. These two data streams may potentially be on different channels, and therefore, may require interface switching (in experiments using three or four channels). However, we see that in spite of interface switching, TCP performance is comparable to UDP (up to three channels).

We have found that TCP performance starts degrading when more than three channels are used. Looking at the TCP traffic received over one second intervals, we see that in certain intervals the nodes receive no packets, implying that TCP timeouts are occurring in experiments with more than three channels. TCP timeouts arise when there are multiple packet losses. We speculate that packet losses are occurring because of cross-channel interference when more than three channels are used. Our interference experiments (described in Chapter 2) suggested that five orthogonal channels may be available for use in our testbed. However, those experiments only considered aggregate throughput and did not consider packet losses. In addition, those results considered interference when traffic was present on only two channels at a time. In this testbed experiment, all nodes are close to each other, and traffic is present on all channels simultaneously, which may

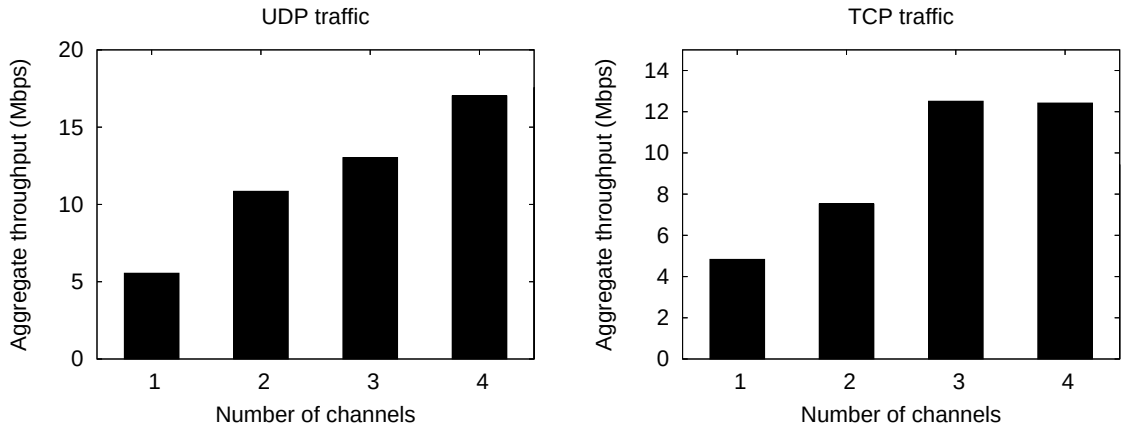

Figure 7.9: Throughput in a single hop network with varying number of channels. Channel data rate is 6 Mbps.

lead to increased cross channel interference. UDP does not slow down in the face of packet losses. Therefore, UDP throughput continues to improve even if a few packets are lost because of cross-channel interference.

7.5.2 Multihop experiments

The performance of the multichannel implementation under multihop scenarios is measured by setting up a chain of five nodes. A flow (TCP or UDP) is set up from the first node in the chain to the last node in the chain (i.e., a 4-hop flow is set up). The flow throughput is measured while the number of channels is varied from one to four. Channel data rate is set to 6 Mbps.

Figure 7.10 plots the flow throughput for both TCP and UDP traffic. As we can see from the figure, when more channels are available, there is a substantial improvement in the throughput, though the magnitude of improvement is less with TCP traffic. TCP throughput depends on the end-to-end delay in addition to the available bandwidth. The interface switching delay is 5 ms with our testbed nodes, and the delay experienced by packets could be higher because of the queuing introduced by the channel abstraction layer. We suspect that this increased delay could be limiting the performance improvements. Note that although a single TCP flow may not utilize all the available bandwidth, other flows in the vicinity could still utilize the unused bandwidth. Therefore, the aggregate network throughput is still expected to increase even with multihop traffic. Similar

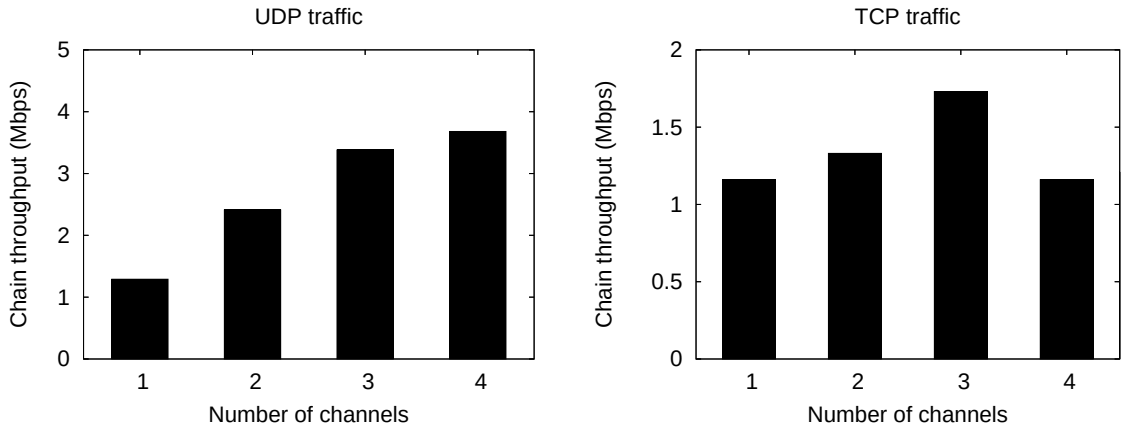

Figure 7.10: Throughput in a 4-hop chain topology with varying number of channels. Channel data rate is 6 Mbps.

to one hop experiments, TCP throughput in a chain topology is also lower with more than three channels. As before, we speculate that this is because of the increased packet losses arising out of cross channel interference.

7.6 Discussions

In this chapter, we described the implementation of the protocols proposed in this thesis in a Linux-based testbed. The prototype implementation has demonstrated the feasibility of using the proposed protocols in practice. As part of the implementation, we identified a generic channel abstraction architecture to support multichannel protocols. The architecture and the protocol suite that we have developed may be useful to develop other multichannel protocols as well.

The channel abstraction architecture was designed to support frequent interface switching. Several commercially available radios now export a rich set of features to the user, such as the ability to set the data rate and transmission power on a per-packet basis. However, to benefit from these features, new higher layer protocols may be required. The implementation of higher layer protocols that utilize these features will require new kernel support as well, and the channel abstraction layer can be easily extended to provide the desired support. We believe that the channel abstraction layer design can serve as a blueprint for building support for any of the features exported by the radio interface.

CHAPTER 8

CONCLUSIONS

In this thesis, we have investigated the use of multiple channels to improve the network performance in multihop wireless networks. The thesis is directed at efficiently using the available frequency spectrum, as well as being flexible enough to incorporate new spectrum as and when it becomes available. The thesis included a study on the capacity of multichannel wireless networks, design of a protocol suite for using multiple channels, a testbed implementation, and simulation-based evaluation of the proposed protocols.

The capacity analysis showed that a large number of channels can be fully utilized even if each node has a only a few interfaces. This result is encouraging, as it implies that if more frequency spectrum becomes available, then the additional frequency can potentially be used without requiring extra hardware. The capacity analysis initially assumed that interfaces could switch instantaneously, but later extended the results to the case when interface switching delay is non-negligible. We showed that even with non-negligible switching delay, there is no loss in the network capacity if the number of interfaces per node is increased. We also considered the scenario where interfaces are not allowed to switch at all, and showed that if all nodes have only one interface, then there is a degradation in network capacity. However, this degradation could be avoided by equipping each node with only two interfaces. These results allow for substituting an (expensive) interface having a small switching delay, with several (inexpensive) interfaces with larger switching delays, without any loss in the asymptotic capacity. The insights from capacity analysis have been used to develop the architecture and protocols for multichannel networks.

New protocols were developed to utilize multiple channels and multiple interfaces. The protocols we designed are capable of operating with off-the-shelf hardware. We separated the protocol design

into interface management and routing protocols for reducing cross-layer interactions. We proposed an interface assignment strategy that keeps one interface at each node fixed on a channel, while the second interface can switch among the remaining channels. Our proposal improved over past work by being easy to implement, while at the same time effectively utilizing all the available channels. We also presented a distributed protocol for selecting fixed channels, and a detailed evaluation of the interface management protocol. After identifying the shortcomings of some of the existing metrics, we presented a new routing metric for multichannel networks. The new metric improved over past proposals by choosing channel diverse routes, while incorporating the impact of interface switching latency. The routing metric was included in an on-demand routing protocol, and detailed evaluations were presented. The protocol evaluations have demonstrated that significant performance improvements is possible even if each node has only two interfaces.

Implementing the protocols in a testbed required new operating system support. We identified the features needed in operating systems to support protocols that require interface switching. We addressed this need by developing a new abstraction layer for managing multiple channels and multiple interfaces. The protocols that we have proposed have been implemented using the new abstraction layer. The implementation demonstrated the practical feasibility of the proposed multichannel protocols, and may serve as a starting point for future multichannel implementations.

In summary, the thesis has comprehensively studied the benefits of using multiple channels, and presented a fully implemented protocol suite for extracting the expected performance benefits in realistic scenarios.

There are several avenues for future work. The capacity analysis could be extended to better understand the impact of switching delay on capacity. It is an open problem to determine if switching delay reduces the network capacity when a single interface is available. All the capacity results are derived under an asymptotic setting. There are several recent efforts that study the network capacity with a small number of nodes, but those efforts do not consider the impact of switching delay as well. Since switching delay may be non-negligible in the future as well (because of protocol delays after a physical switch, for example, to update virtual carrier sense information), understanding its impact is an important problem.

Driven by the need to simplify protocols, we have not incorporated some of the insights from capacity analysis in the proposed protocols. For example, we do not explicitly balance flows across nodes, and we do not choose the transmission power based on the number of channels. It is an open problem to incorporate those insights into practical protocols. Protocol design has also primarily focused on homogeneous channels, although the protocols are robust in the face of channel heterogeneity. There is a growing interest in using smart radios that operate over a wide range of frequency bands, and channel heterogeneity is expected to be common in such a scenario. Therefore, it is an interesting problem to design protocols that exploit channel heterogeneity.

Motivated by the lack of sufficient operating system support, we have developed the new channel abstraction layer. Although our implementation focused on the use of multiple channels, the abstraction layer can be easily extended to support multiple data rates, multiple transmission powers, directional antennas, etc. There is a growing trend toward exposing more of the radio features to higher layers, and new protocols are required to exploit these radio features. Our multichannel implementation may serve as a template for building solutions that fully exploit the features exposed by the radio.

REFERENCES

[1] *IEEE Standard for Wireless LAN-Medium Access Control and Physical Layer Specification, P802.11*, 1999.

[2] Kenneth R. Carter, "Openness and the Public Airwaves," Panel Discussion at Mobihoc, 2005.

[3] Paramvir Bahl, Atul Adya, Jitendra Padhye, and Alec Wolman, "Reconsidering Wireless Systems with Multiple Radios," *ACM Computing Communication Review*, July 2004.

[4] Richard Draves, Jitendra Padhye, and Brian Zill, "Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks," in *ACM Mobicom*, 2004.

[5] Arunesh Mishra, Vivek Shrivastava, Suman Banerjee, and William Arbaugh, "Partially-overlapped Channels not considered harmful," in *ACM Sigmetrics*, St. Malo, France, Jun 2006.

[6] Chinh H. Doan, Sohrab Emami, David A. Sobel, Ali M. Niknejad, and Robert W. Brodersen, "Design Considerations for 60 GHz CMOS Radios," *IEEE Communications Magazine*, vol. 42, no. 132-140, Dec 2004.

[7] Johnna Powell, "Antenna Design for Ultra Wideband Radio," M.S. thesis, Massachusetts Institute of Technology, 2004.

[8] Puneet Prashant Newaskar, "High-Speed Data Conversion for Digital Ultra-Wideband Radio Receivers," M.S. thesis, Massachusetts Institute of Technology, 2003.

[9] Xue Yang, *Efficient Packet Scheduling in Wireless Multihop Networks*, Ph.D. thesis, University of Illinois at Urbana-Champaign, 2005.

[10] "Net 4521 hardware from Soekris," http://www.soekris.com/net4521.htm.

[11] "Atheros Inc," http://www.atheros.com.

[12] Chandrakanth Chereddi, "System architecture for multichannel multi-interface wireless networks," M.S. thesis, University of Illinois at Urbana-Champaign, 2006.

[13] "Multiband Atheros Driver for WiFi (MADWiFi), BSD-branch CVS snaphot," May 25th, 2005, http://madwifi.org.

[14] Piyush Gupta and P. R. Kumar, "The Capacity of Wireless Networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, March 2000.

[15] Matthias Grossglauser and David Tse, "Mobility Increases the Capacity of Ad-hoc Wireless Networks," in *Infocom*, 2001.

[16] Pradeep Kyasanur and Nitin H. Vaidya, "Capacity of multi-channel wireless networks: impact of number of channels and interfaces," in *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*, New York, NY, USA, 2005, pp. 43–57, ACM Press.

[17] Abbas El Gamal, James Mammen, Balaji Prabhakar, and Devavrat Shah, "Throughput-Delay Trade-off in Wireless Networks," in *Infocom*, 2004.

[18] Nikhil Bansal and Zhen Liu, "Capacity, Delay and Mobility in Wireless Ad-Hoc Networks," in *Infocom*, 2003, pp. 1553–1563.

[19] Eugene Perevalov and Rick Blum, "Delay Limited Capacity of Ad hoc Networks: Asymptotically Optimal Transmission and Relaying Strategy," in *Infocom*, 2003.

[20] Xiaojun Lin and Ness B. Shroff, "The Fundamental Capacity-Delay Tradeoff in Large Mobile Ad Hoc Networks," Tech. Rep., Purdue University, 2004.

[21] Xiaojun Lin, Gaurav Sharma, Ravi R. Mazumdar, and Ness B. Shroff, "Degenerate Delay-Capacity Trade-offs in Ad Hoc Networks with Brownian Mobility," *Joint Special Issue of IEEE Transactions on Information Theory and IEEE/ACM Transactions on Networking on Networking and Information Theory*, vol. 52, no. 6, pp. 2777–2784, June 2006.

[22] Benyuan Liu, Zhen Liu, and Don Towsley, "On the Capacity of Hybrid Wireless Networks," in *Infocom*, 2003.

[23] Ulas C. Kozat and Leandros Tassiulas, "Throughput Capacity of Random Ad Hoc Networks with Infrastructure Support," in *Mobicom*, 2003.

[24] A. Agarwal and P. R. Kumar, "Capacity bounds for ad-hoc and hybrid wireless networks," *ACM SIGCOMM CCR*, vol. 34, no. 3, pp. 71–81, July 2004.

[25] Su Yi, Yong Pei, and Shivkumar Kalyanaraman, "On the Capacity Improvement of Ad Hoc Wireless Networks Using Directional Antennas," in *Mobihoc*, USA, 2003, pp. 108–116.

[26] Rohit Negi and Arjunan Rajeswaran, "Capacity of power constrained ad-hoc networks," in *Infocom*, 2004.

[27] Honghai Zhang and Jennifer C. Hou, "Capacity of Wireless Ad-hoc Networks under Ultra Wide Band with Power Constraint," in *Infocom*, 2005.

[28] Jiandong Li, Zygmunt J. Haas, , and Min Sheng, "Capacity Evaluation of Multi-Channel Multi-Hop Ad Hoc Networks," in *International Conference on Personal Communications*, 2002.

[29] Michael Gastpar and Martin Vetterli, "On the Capacity of Wireless Networks: The Relay Case," in *Infocom*, New York, USA, June 2002, pp. 1577–1586.

[30] Stavros Toumpis and Andrea J. Goldsmith, "Large Wireless Networks under Fading, Mobility, and Delay Constraints," in *Infocom*, 2004.

[31] Stavros Toumpis, "Capacity Bounds for Three Classes of Wireless Networks: Asymmetric, Cluster, and Hybrid," in *Mobihoc*, 2004.

[32] M. Kodialam and T. Nandagopal, "Characterizing the Capacity Region in Multi-Radio Multi-Channel Wireless Mesh Networks," in *ACM Mobicom*, 2005.

[33] M. Alicherry, R. Bhatia, and L. Li, "Joint Channel Assignment and Routing for Throughput Optimization in Multi-radio Wireless Mesh Networks," in *ACM Mobicom*, 2005.

[34] Jihui Zhang, Haitao Wu, Qian Zhang, and Bo Li, "Joint routing and scheduling in multi-radio multi-channel multi-hop wireless networks," in *2nd International Conference on Broadband Networks*, Oct 2005, vol. 1, pp. 631–640.

[35] Ashish Raniwala and Tzi-cker Chiueh, "Architecture and Algorithms for an IEEE 802.11-Based Multi-Channel Wireless Mesh Network," in *Infocom*, 2005.

[36] Pradeep Kyasanur and Nitin H. Vaidya, "Routing and Interface Assignment in Multi-Channel Multi-Interface Wireless Networks," in *WCNC*, 2005.

[37] Martin Raab and Angelika Steger, ""Balls into Bins" - A Simple and Tight Analysis," in *2nd Workshop on Randomization and Approximation Techniques in Computer Science*, 1998.

[38] S. Diggavi, M. Grossglauser, and D. Tse, "Even One-Dimensional Mobility Increases Adhoc Wireless Capacity," Tech. Rep., UC Berkeley, 2003.

[39] V. N. Vapnik, *Estimation of Dependences based on Empirical Data*, Springer-Verlag, New York, 1982.

[40] V. N. Vapnik and A. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and its Applications*, vol. 16, no. 2, pp. 264–280, 1971.

[41] Douglas B. West, *Introduction to Graph Theory*, Prentice Hall, second edition, 2001.

[42] Vartika Bhandari and Nitin H. Vaidya, "Connectivity and Capacity of Multi-Channel Wireless Networks with Channel Switching Constraints," Tech. Rep., University of Illinois at Urbana-Champaign, August 2006.

[43] Donald E. Knuth, *The art of computer programming, fundamental algorithms*, vol. 1, chapter 1.3.3, Addison Wesley, 3 edition, 1997.

[44] "Notes on permutation cycles," http://www.inference.phy.cam.ac.uk/mackay/itila/cycles.pdf.

[45] Bela Bollobas, *Random Graphs*, Harcourt Brace Jovanovich, 1985.

[46] A. Nasipuri, J. Zhuang, and S.R. Das, "A Multichannel CSMA MAC Protocol for Multihop Wireless Networks," in *WCNC*, September 1999.

[47] A. Nasipuri and S.R. Das, "Multichannel CSMA with Signal Power-based Channel Selection for Multihop Wireless Networks," in *VTC*, September 2000.

[48] N. Jain, S. Das, and A. Nasipuri, "A Multichannel CSMA MAC Protocol with Receiver-Based Channel Selection for Multihop Wireless Networks," in *IEEE International Conference on Computer Communications and Networks (IC3N)*, October 2001.

[49] Atul Adya, Paramvir Bahl, Jitendra Padhye, Alec Wolman, and Lidong Zhou, "A Multi-Radio Unification Protocol for IEEE 802.11 Wireless Networks," in *IEEE International Conference on Broadband Networks (Broadnets)*, 2004.

[50] Shih-Lin Wu, Chih-Yu Lin, Yu-Chee Tseng, and Jang-Ping Sheu, "A New Multi-Channel MAC Protocol with On-Demand Channel Assignment for Multi-Hop Mobile Ad Hoc Networks," in *International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, 2000.

[51] Priya Ravichandran, "Explicitly Pipelining IEEE 802.11 to Enhance Performance," M.S. thesis, University of Illinois at Urbana-Champaign, December 2003.

[52] Wing-Chung Hung, K.L.Eddie Law, and A. Leon-Garcia, "A Dynamic Multi-Channel MAC for Ad Hoc LAN," in *21st Biennial Symposium on Communications*, Kingston, Canada, June 2002, pp. 31–35.

[53] Jenhui Chen, Shiann-Tsong Sheu, and Chin-An Yang, "A New Multichannel Access Protocol for IEEE 802.11 Ad Hoc Wireless LAN," in *14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communication*, 2003.

[54] Krishna N. Ramachandran, Elizabeth M. Belding, Kevin C. Almeroth, and Milind M. Buddhikot, "Interference-Aware Channel Assignment in Multi-Radio Wireless Mesh Networks," in *Infocom*, Barcelona, Spain, Apr 2006.

[55] M. X. Gong and S. F. Midkiff, "Distributed Channel Assignment Protocols: A Cross-Layer Approach," in *IEEE Wireless Communications and Networking Conference (WCNC)*, March 2005.

[56] M. X. Gong, S. F. Midkiff, and S. Mao, "A combined proactive routing and multi-channel MAC protocol for wireless ad hoc networks," in *2nd International Conference on Broadband Networks*, Boston, MA, Oct 2005, vol. 1, pp. 444–453.

[57] Jungmin So and Nitin H. Vaidya, "Multi-channel MAC for Ad Hoc Networks: Handling Multi-Channel Hidden Terminals using a Single Transceiver," in *Mobihoc*, 2004.

[58] Jungmin So, *Design and Evaluation of Multi-Channel Multi-Hop Wireless Networks*, Ph.D. thesis, University of Illinois at Urbana-Champaign, 2006.

[59] P. Bahl, R. Chandra, and J. Dunagan, "SSCH: Slotted Seeded Channel Hopping for Capacity Improvement in IEEE 802.11 Ad-Hoc Wireless Networks," in *ACM Mobicom*, 2004.

[60] Marco Ajmone Marsan and Fabio Neri, "A Simulation Study of Delay in Multichannel CSMA/CD Protocols," *IEEE Transactions on Communications*, vol. 39, no. 11, pp. 1590–1603, November 1991.

[61] Jian Tang, Guoliang Xue, and Weiyi Zhang, "Interference-aware topology control and QoS routing in multi-channel wireless mesh networks," in *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, New York, NY, USA, 2005, pp. 68–77, ACM Press.

[62] David B. Johnson, David A. Maltz, and Yih-Chun Hu, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)," *Ietf Manet Working Group (Draft 10)*, 2004.

[63] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," in *Ietf RFC 3561*, July 2003.

[64] N. Shacham and P. King., "Architectures and Performance of Multichannel Multihop Packet Radio Networks," *IEEE Journal on Selected Area in Communications*, vol. 5, no. 6, pp. 1013–1025, July 1987.

[65] Jungmin So and Nitin H. Vaidya, "A Routing Protocol for Utilizing Multiple Channels in Multi-Hop Wireless Networks with a Single Transceiver," Tech. Rep., University of Illinois at Urbana-Champaign, October 2004.

[66] Ashish Raniwala, Kartik Gopalan, and Tzi-cker Chiueh, "Centralized Channel Assignment and Routing Algorithms for Multi-Channel Wireless Mesh Networks," *Mobile Computing and Communications Review*, vol. 8, no. 2, pp. 50–65, April 2004.

[67] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," in *ACM Mobicom*, 2003.

[68] Pradeep Kyasanur, Jitendra Padhye, and Paramvir Bahl, "A Study of an 802.11-like Control Channel-Based MAC," in *IEEE International Conference on Broadband Networks (Broadnets)*, 2005.

[69] Pradeep Kyasanur and Nitin H. Vaidya, "Routing and Link-layer Protocols for Multi-Channel Multi-Interface Ad hoc Wireless Networks," *Sigmobile Mobile Computing and Communications Review*, vol. 10, no. 1, pp. 31–43, Jan 2006.

[70] Pradeep Kyasanur and Nitin H. Vaidya, "Protocol Design Challenges for Multi-hop Dynamic Spectrum Access Networks," in *IEEE DySpan*, 2005.

[71] Pradeep Kyasanur, Xue Yang, and Nitin H. Vaidya, "Mesh Networking Protocols to Exploit Physical Layer Capabilities," in *IEEE Workshop on Wireless Mesh Networks (Wimesh)*, 2005.

[72] Scalable Network Technologies, "Qualnet simulator version 3.9," http://www.scalable-networks.com.

[73] Gavin Holland and Nitin Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, New York, NY, USA, 1999, pp. 219–230, ACM Press.

[74] Yaling Yang, Jun Wang, and Robin Kravets, "Designing Routing Metrics for Mesh Networks," in *IEEE Workshop on Wireless Mesh Networks (WiMesh)*, 2005.

[75] D. Maltz, J. Broch, and D. Johnson, "Experiences Designing and Building a Multi-Hop Wireless Ad-Hoc Network Testbed," Tech. Rep. TR99-116, CMU, 1999.

[76] H. Lunndgren, D. Lundberg, J. Nielsen, E. Nordstrom, and C. Tscudin, "A Large-scale Testbed for Reproducible Ad Hoc Protocol Evaluations," in *WCNC*, 2002.

[77] B. A. Chambers, "The Grid Roofnet: A Rooftop Ad Hoc Wireless Network," M.S. thesis, MIT, 2002.

[78] R. Karrer, A. Sabharwal, and E. Knightly, "Enabling Large-scale Wireless Broadband: The Case for TAPs," in *Hotnets*, 2003.

[79] B. White, J. Lepreau, and S. Guruprasad, "Lowering the Barrier to Wireless and Mobile Experimentation," in *Hotnets*, 2002.

[80] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols," in *WCNC*, 2005.

[81] P. De, A. Raniwala, S. Sharma, and T. Chiueh, "MiNT: A Miniaturized Network Testbed for Mobile Wireless Research," in *Infocom*, 2005.

[82] Ranveer Chandra, Paramvir Bahl, and Pradeep Bahl, "MultiNet: Connecting to Multiple IEEE 802.11 Networks Using a Single Wireless Card," in *IEEE Infocom*, Hong Kong, March 2004.

[83] "Virtual Wifi software page," http://research.microsoft.com/netres/projects/virtualwifi.

[84] Patrick Stuedi and Gustavo Alonso, "Transparent Heterogeneous Mobile Ad Hoc Networks," in *ACM MobiQuitous*, 2005.

[85] Nicolas Boulicault, Guillaume Chelius, and Eric Fleury, "Experiments of Ana4: An Implementation of a 2.5 Framework for Deploying Real Multi-hop Ad-hoc and Mesh Networks," in *REALMAN*, 2005.

[86] Vikas Kawadia, Yongguang Zhang, and Binita Gupta, "System Services for Implementing Ad-Hoc Routing: Architecture, Implementation and Experiences," in *Mobisys*, 2003.

[87] "Netfilter: Linux packet filtering framework," http://www.netfilter.org/.

[88] "AODV-UU: AODV implementation from Uppsala University, version 0.9.1," http://core.it.uu.se/AdHoc/ImplementationPortal.

[89] Matthew J. Miller, William D. List, and Nitin H. Vaidya, "A Hybrid Network Implementation to Extend Infrastructure Reach," Tech. Rep., University of Illinois at Urbana-Champaign, 2003.

[90] "Champaign-Urbana Community Wireless Network," http://www.cuwireless.net/.

[91] "Seattle Wireless," http://www.seattlewireless.net/.

[92] "Municipal Wireless," http://www.muniwireless.com/.

[93] "Iperf version 2.0.2," May 2005, http://dast.nlanr.net/Projects/Iperf/.

AUTHOR'S BIOGRAPHY

Pradeep Kyasanur was born in Karnataka, India in 1979. Pradeep received the BE degree in Computer Engineering from Mangalore University, India, in 2001. He was awarded the first rank in computer engineering by the Mangalore University for his undergraduate academic performance. In August 2001, Pradeep joined the Computer Science department at the University of Illinois at Urbana-Champaign. Pradeep received an MS degree in Computer Science from University of Illinois at Urbana-Champaign in 2003 under the guidance of Prof. Nitin H. Vaidya. His MS thesis was entitled *Selfish Misbehavior at Medium Access Control Layer in Wireless Networks*. Pradeep's graduate work has been supported by a Vodafone Gradate Fellowship during 2004-2006. Pradeep is a student member of the IEEE.