DESIGN AND EVALUATION OF MULTI-CHANNEL MULTI-HOP WIRELESS NETWORKS

BY

JUNGMIN SO

B.S., Seoul National University, 2001

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

DESIGN AND EVALUATION OF MULTI-CHANNEL MULTI-HOP WIRELESS NETWORKS

Jungmin So, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 2006
Nitin H. Vaidya, Adviser

Wireless networks have been widely deployed in the past few years. With rapidly increasing number of users, it is important to maximize utilization of given resources to achieve high performance. Among many approaches, the aim of this dissertation is in utilizing available frequency spectrum, which may be divided into multiple non-overlapping channels. Particularly, we focus on the case where wireless nodes are equipped with a single network interface, which can operate on any one channel at a given time.

Wireless network standards such as IEEE 802.11 provide multiple non-interfering frequency channels, where multiple communications can take place simultaneously. However, current protocols cannot exploit this availability to achieve high performance. The main challenge in utilizing multiple channels is the fact that a node equipped with a single network interface can only operate on a single channel, although it is capable of switching its operating channel. When two nodes want to communicate, they must be operating on the same channel. Thus, to support multiple channels, protocols need to properly assign and coordinate channels among nodes. Without careful design, multi-channel protocols may not be able to achieve higher performance than single-channel protocols.

In this disseration, we first identify issues in designing a multi-channel multi-hop network, with the focus on layer-2 and layer-3 protocols. Based on the observations, we develop medium access control (MAC) and routing protocols that can utilize multiple channels with a single interface. Finally, we evaluate the multi-channel protocols through simulation and a prototype implementation.

Abstract

Wireless networks have been widely deployed in the past few years. With rapidly increasing number of users, it is important to maximize utilization of given resources to achieve high performance. Among many approaches, the aim of this dissertation is in utilizing available frequency spectrum, which may be divided into multiple non-overlapping channels. Particularly, we focus on the case where wireless nodes are equipped with a single network interface, which can operate on any one channel at a given time.

Wireless network standards such as IEEE 802.11 provide multiple non-interfering frequency channels, where multiple communications can take place simultaneously. However, current protocols cannot exploit this availability to achieve high performance. The main challenge in utilizing multiple channels is the fact that a node equipped with a single network interface can only operate on a single channel, although it is capable of switching its operating channel. When two nodes want to communicate, they must be operating on the same channel. Thus, to support multiple channels, protocols need to properly assign and coordinate channels among nodes. Without careful design, multi-channel protocols may not be able to achieve higher performance than single-channel protocols.

In this disseration, we first identify issues in designing a multi-channel multi-hop network, with the focus on layer-2 and layer-3 protocols. Based on the observations, we develop medium access control (MAC) and routing protocols that can utilize multiple channels with a single interface. Finally, we evaluate the multi-channel protocols through simulation and a prototype implementation.

To Mom and Dad

Acknowledgments

Looking back, I am truly grateful that I met so many wonderful people in my life journey, and without them the completion of this dissertation would not have been possible.

First of all, I would like to express my gratitude to my adviser, Prof. Nitin Vaidya. When I first entered the Ph.D. program, I had many difficulties getting used to life as a graduate student. Especially, I had no previous experience in doing research, so it took me some time before I could think like a researcher. Throughout all the difficult times I had, Prof. Vaidya was always understanding and guided me with patience. Discussions I had with him were very enjoyable, and greatly inspired me in an encouraging way.

I would like to thank all my committee members, Prof. Gul Agha, Prof. Klara Nahrstedt, and Prof. Haiyun Luo. Their comments and suggestsions were invaluable in making this thesis complete. I would like to thank Prof. Geneva Belford, for her kind support and advice throughout the grduate years. I would like to thank all my research group members, Romit Roy Choudhury, Xue Yang, Pradeep Kyasanur, Matthew Miller, Vartika Bhandari, Chandrakanth Chereddi, and Wonyong Yoon, for all the support they have given me and the contributions they have made for this thesis. I am grateful that I could join this research group and work with this wonderful group of people. I would like to thank Korea Foundation for Advanced Studies and Motorola Center for Communications for providing financial support.

My life at graduate school was not just about research, but learning to become a mature person both intellectually and emotionally. So many people have given me support as I struggled to overcome many new challenges. I would like to thank the cool guys, Jintae Kim, Yongmin Ko, and Eun-Gyu Kim. We had so many things to share as men, graduate students, computer scientists, and Christians. I am thankful for each one of them for being with me in every situation. I thank all my church members for their kind support. I especially thank Seunghee Ha for all the comfort she has given me during my studies.

Finally, I would like to thank my father, Hyunsoo So, my mother, Sueok Shin, and my sister, Jungwon So, for the unconditional love they have given me. Without them I would definitely not have come this far. Above everything, I believe that God has led me all the way to this point, and I am thankful for everything He has provided me during my graduate years.

Table of Contents

Chapter 1

Introduction

In recent years, the number of wireless network users have been rapidly increasing. Keeping pace with demand, more and more wireless access points are being deployed in homes and public buildings such as airports and cafes to provide wireless access. The commercial deployment of wireless networks has been made possible by standards such as IEEE 802.11 [5].

As the popularity of wireless networks increases, certain places become "hot-spots" where a large number of users in the same area connect to the Internet via wireless channels. In those areas, it becomes very important to utilize available bandwidth efficiently to achieve high performance.

Wireless standards such as IEEE 802.11 provide multiple frequency channels available for use. For example, IEEE 802.11b defines 14 channels, numbered from 1 to 14. However, not all channels can be used simultaneously, because channels close to each other in frequency band can interfere with each other. In general, channels 1, 6, and 11 can be used for communication simultaneously. With IEEE 802.11a, 12 non-overlapping channels are available.

Even if there are multiple channels, a single network interface can only operate on one channel at a time, although it can switch channels at the cost of channel switching delay. In order to make use of multiple channels, a node must have multiple network interfaces, or it must divide time to access multiple channels one at a time.

Most of current mobile devices and access points (AP) are equipped with a single network interface. In wireless LANs, an AP is assigned a fixed operating channel by manual configuration, and all mobile devices in the range of the AP communicates with the AP on that fixed channel. When multiple APs are deployed to cover a large area, neighboring APs typically use different channels to avoid interference in the overlapping regions. This is illustrated in Figure 1.1.

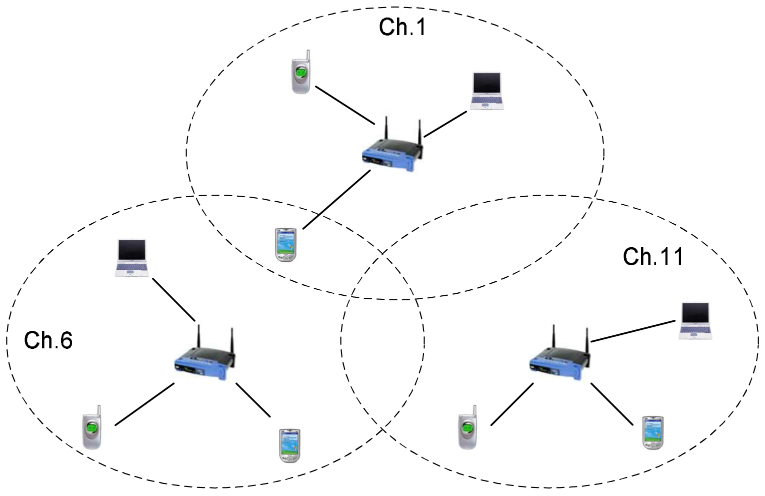


Figure 1.1: An example deployment of wireless LAN access points.

Infrastructure-based wireless LANs are networks that provide connection to the mobile devices directly within reach of an access point connected to the wired network. There are also other types of wireless networks that have been proposed. For example, ad hoc networks [6] are wireless networks without infrastructure. An ad hoc network can be established by mobile devices directly connecting to other mobile devices within the radio range. An ad hoc network often allows multi-hop connection, where packets travel multiple wireless hops from sender to the receiver via other mobile devices. An example ad hoc network is shown in Figure 1.2.

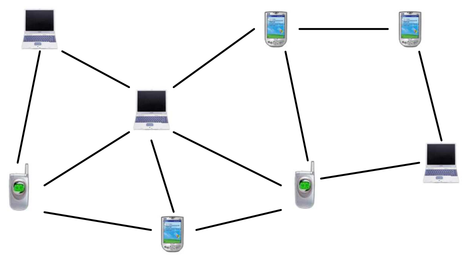


Figure 1.2: An example ad hoc networks.

Similar to ad hoc networks, mesh networks [7–10] are network of wireless routers (also called mesh routers) connected through wireless interfaces. Hybrid networks [11–14] are networks where infrastructure exists, but mobile devices may also connect with each other, directly or via other mobile devices.

The focus of this dissertation is to develop protocols for multi-hop networks that utilize multiple channels to achieve high performance, under the constraint that each node is equipped with a single interface. It is not trivial to use multiple non-overlapping channels under the single-interface constraint, because of the fundamental reason that two nodes on different channels cannot hear each other. If nodes are equipped with multiple interfaces, it may make the protocol simpler or more efficient in terms of channel utilization. However, it may be too expensive to equip multiple network interfaces into low-cost devices such as phones and PDAs. We limit our scope of research to protocols that support devices with single interface. Protocols for devices with multiple interfaces [15–17] can be applied to mesh networks, where hardware cost may not be as critical a factor.

One might argue that it is better to combine all the available frequency spectrum and use it as a single wide-band channel. There can be several advantages of using a single wide-band channel. First, if a frequency band is divided into multiple channels, some bandwidth must be used as "guard bands" to prevent interference between adjacent channels. A single wide-band channel would not need guard bands so it can make use of more bandwidth. Second, the protocols can be simpler because they do not need to address issues with using multiple channels. (These issues will be discussed throughout the dissertation.) However, there are motivations for using multiple channels instead of a single wide-band channel. First, available frequency spectrum may not be continuous. In this case, it is not possible to combine the whole spectrum into a single channel. Even if each continuous frequency spectrum is made into a single channel, we are left with multiple channels. Second, a wide-band channel may require sophiscated hardware with high-performance signal processing capability. For applications that do not require high-rate communication, hardware cost may be unnecessarily expensive. Finally, some protocols require rate-independent overhead, as discussed in [72]. The backoff mechanism in IEEE 802.11 DCF [5] is an example of rate-independent overhead, which does not depend on the rate. For rate-independent overhead, bandwidth loss due to the overhead becomes higher when the channel bit-rate increases. Dividing frequency spectrum into multiple channels will reduce the bandwith loss caused by rate-independent overhead.

The protocols of interest in this dissertation are MAC and routing protocols. Multi-channel MAC protocols and multi-channel routing protocols have the same goal of exploiting multiple channels to improve network performance, but dealing with multiple channels at different layers [1]

[1]MAC and routing protocols fall into layer 2 and layer 3 in OSI 7-layer model, respectively.

has different advantages and disadvantages. For example, the MAC layer only has local knowledge, but it has more up-to-date information on channel quality. On the other hand, network layer may have larger view of the network, but it takes more time to gain information on channel quality and respond to current situation. This difference can affect the protocols (such as timescale of channel assignment).

The major contributions of this disseration are as follows. First, we study issues in multi-channel networks, and identify problems with conventional MAC and routing protocols when applied to multi-channel networks. Second, we propose various approaches for layer-2 (link) and layer-3 (network) protocols to address the issues and utilize multiple channels effectively. Third, based on the proposed approaches, we develop MAC and routing protocols for multi-channel networks in the context of ad hoc and hybrid networks. Finally, we evaluate the protocols using simulations and a prototype implemenation to study their effectiveness. The overview of the contributions in accordance to the OSI 7-layer model is depicted in Figure 1.3.

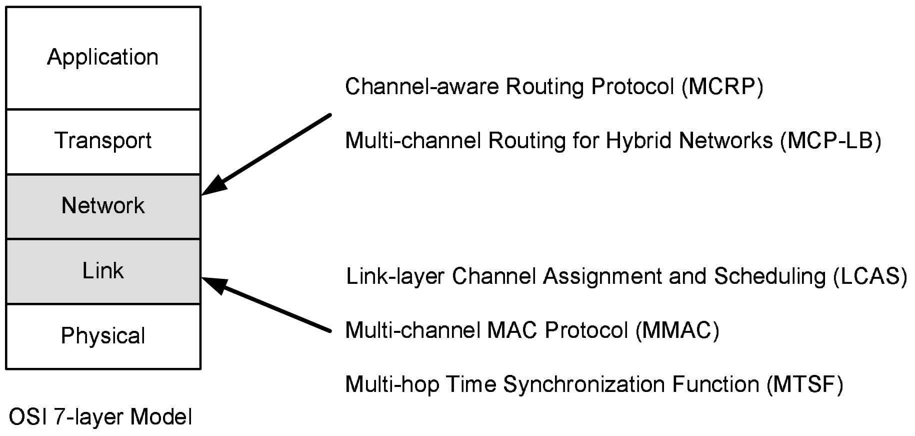

Application
Transport
Network
Link
Physical
OSI 7-layer Model
Channel-aware Routing Protocol (MCRP)
Multi-channel Routing for Hybrid Networks (MCP-LB)
Link-layer Channel Assignment and Scheduling (LCAS)
Multi-channel MAC Protocol (MMAC)
Multi-hop Time Synchronization Function (MTSF)

Figure 1.3: Overview of contributions of the dissertation.

Outline of the Dissertation: Background on protocols for wireless networks are presented in Chapter 2. In Chapter 3, we discuss MAC layer issues in multi-channel networks, and propose a multi-channel MAC protocol called MMAC, which uses periodic channel negotiation. Motivated by synchronous operations in MMAC, in Chapter 4 we develop a light-weight clock synchronization algorithm to support synchronous operation in multi-hop networks. In Chapter 5, we integrate MMAC with clock synchronization, and also implement an extension to AODV routing protocol

[18] that can produce better performance when working with MMAC. In Chapter 6, we discuss network layer issues and develop a multi-channel routing protocol for ad hoc networks called MCRP. As discussed in Chapters 5 and 6, MMAC and MCRP can assign different channels to different flows, but still they cannot exploit multiple channels within a flow. In Chapter 7, we go further and develop a link-layer channel assignment algorithm that can utilize multiple channels within a flow. In Chapter 8, we look at hybrid networks, where all traffic is sent to the access points, and develop a multi-channel routing protocol in the context of hybrid networks. Since hybrid networks have simpler traffic patterns than ad hoc networks, the protocol becomes simpler in terms of route management and focuses more on load balancing. In Chapter 9, we describe our prototype implementation, and discuss issues regarding the implementation. Finally, Chapter 10 concludes the dissertation with possible directions for future research.

Chapter 2

Background

In this chapter, we introduce the basics of medium access control and routing in multi-hop wireless networks. In particular, we discuss IEEE 802.11 Distributed Coordination Function (DCF) [5] for the MAC protocols, and Ad-hoc On-demand Distance Vector protocol (AODV) [18] for the routing protocol. These protocols are well-studied and popularly used in multi-hop wireless networks. Although these protocols are designed for single-channel networks, many concepts and ideas from these protocols will be used as building blocks in developing protocols for multi-channel networks.

2.1 Medium Access Control in Wireless Networks

A wireless channel is a broadcast medium. When a node transmits a packet onto the channel, the signal reaches all nodes in the transmission range of the sender. If a node receives multiple packets on the same channel at the same time, it cannot properly decode the packet because of the interference. In this case, we say that packets have "collided" at the receiver. Thus, the major goal of wireless medium access control (MAC) is to ensure that when a node is transmitting, all other interfering nodes do not transmit.

IEEE 802.11, a widely deployed wireless network standard specifies two MAC protocols, called Point Coordination Function (PCF) and Distributed Coordination Function (DCF) [5]. PCF only works for infrastructure networks, where DCF does not require an infrastructure. DCF is the MAC protocol of our interest here.

DCF is based on a scheme called Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). CSMA/CA can be divided into two parts, CSMA and CA. The basic idea of CSMA is to check whether the channel is busy or idle before transmitting a packet. A node implementing CSMA continuously listens to the channel and measures the signal level on the channel. The node only transmits a packet when the signal level on the channel is sufficiently low, assuming that no other transmission is taking place in the region.

CSMA blocks a node from initiating a transmission while another node nearby is transmitting, but it does not address the following well-known problem called *hidden terminal problem*. Consider the scenario in Figure 2.1. When node A is transmitting a packet to node B, node C may not sense the channel as busy. Thus, node C can start transmitting a packet, which results in collision at node B.

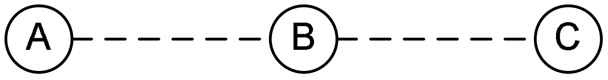


Figure 2.1: A scenario illustrating hidden terminal problem.

Collision avoidance (CA) improves upon CSMA by having a node “reserve” the channel before transmitting a packet. In IEEE 802.11 DCF, CA is implemented using RTS/CTS scheme. In Figure 2.1, suppose node A has a packet to send to B. Before transmitting the data packet, node A sends a RTS (Request-To-Send) packet to B, which contains the duration of time node A needs to reserve. When node B receives RTS, it replies back by sending a CTS (Clear-To-Send) packet, which also includes the duration of time the channel needs to be reserved. All other nodes that receive RTS or CTS defers transmission for the duration specified in the packets. For this reason, each node maintains a variable called the *Network Allocation Vector* (NAV), that records the duration of time the node must defer its transmission. When node C receives CTS from B, it sets up its NAV according to the duration specified in the CTS packet. After exchanging RTS and CTS, node A sends the data frame to node B, and B replies with ACK (Acknowledgement) packet so that node A knows the packet has been successfully received at node B. The whole process of transmitting a data packet using DCF is illustrated in Figure 2.2.

If a node has a packet to send and finds the channel busy, it starts what is called *binary exponential backoff* (BEB). The node selects a backoff counter randomly from the range [0, $CW-1$], where CW is the *contention window*. While the channel is idle, the node counts down the backoff counter after each time slot (a slot is $20\mu s$ duration in IEEE 802.11b). If the channel becomes

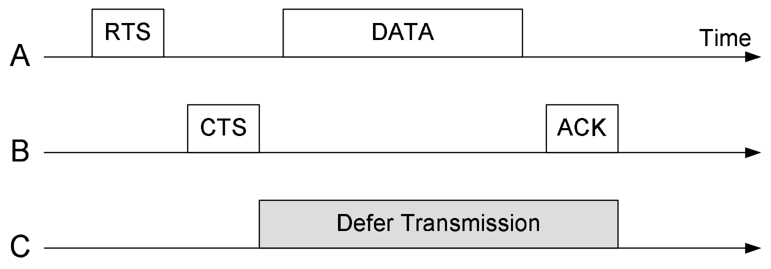


Figure 2.2: A packet transmission procedure using IEEE 802.11 DCF.

busy, the backoff counter is paused until the channel becomes idle again. Each time a transmission is unsuccessful, the node doubles the contention window. If a transmission is successful, it resets the contention window to CW_{min}, a predefined value. The binary exponential backoff mechanism reduces collisions by having nodes initiate transmissions at different times.

2.2 Routing in Multi-hop Wireless Networks

In a multi-hop wireless network, a source node may need to send packets to its destination via multiple intermediate nodes. A routing protocol establishes and maintains paths between nodes. If there are multiple potential paths between two nodes, the routing protocol selects the "best" route based on different metrics (e.g. number of hops, residual energy, minimum delay, channel quality, etc.).

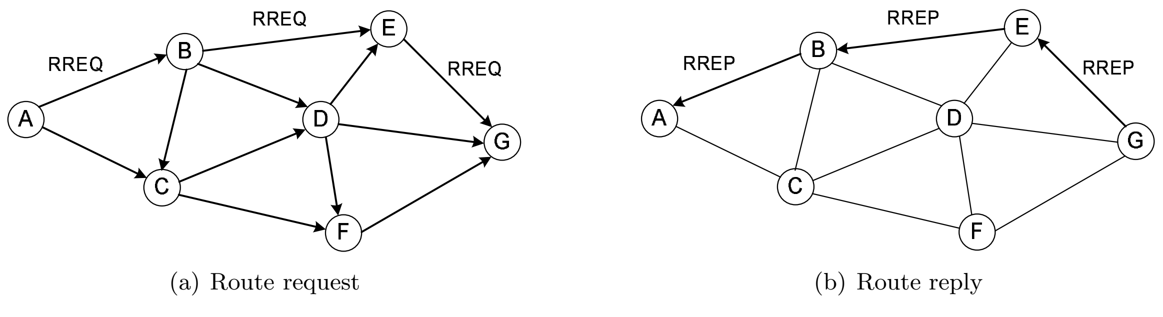
There are numerous routing protocols proposed for multi-hop wireless networks, where all nodes in the network are assumed to be on the same channel [19]. Most of those protocols can be categorized into proactive protocols, reactive protocols and hybrid protocols. Proactive protocols (DSDV [20], OLSR [21]) try to maintain up-to-date routes regardless of whether the route needs to be used. On the other hand, reactive protocols (DSR [22], AODV [18]) establish routes only when there is need. In general, reactive protocols may spend much less cost for route maintenance compared to proactive protocols. However, reactive protocols may require initial delay for route discovery, and often cannot update routes on-the-fly when a better route appears. Hybrid protocols (e.g. ZRP [23]) combine proactive and reactive approaches, such as proactively maintaining routes up to k-hop neighbors, and establishing routes of more than k-hop only when there is need.

Here we describe AODV (Ad-hoc On-demand Distance Vector) [18] protocol in further detail, since similar ideas will be used in developing routing protocols for multi-channel networks. As mentioned above, AODV establishes a path between two nodes only when there is need to transfer packets. A path to a particular destination is described using the following attributes: destination node, next hop, and number of hops. For example, the route table of node A might look like Figure 2.3. From the table, node A knows that in order to reach node G, it has to forward packets to node B, and it will take 3 hops to arrive at node G. Note that AODV prefers a route with short hop distance, so it remembers the number of hops as a measure of route quality. If the desired metric is different, then the route table can record other types of metric for each route.

destination	nexthop	number of hops
G	B	3
F	C	2

Figure 2.3: An example AODV route table.

Route establishment process begins when a source node broadcasts a Route Request (RREQ) packet. Consider the scenario in Figure 2.4. Suppose node A wants to send packets to node G. Since node A does not know a path to G, it broadcasts a RREQ packet indicating it is looking for node G. Each intermediate node forwards the RREQ, and the packet propagates throughout the whole network, as in 2.4(a). Note that each node only forwards RREQ once, and drops all duplicate packets that arrive later. As the RREQ is forwarded, each node sets up a *reverse path* towards the source node A. Thus, all nodes that have received RREQ now have a path to node A.



(a) Route request (b) Route reply

Figure 2.4: A multi-hop network scenario illustrating route discovery process in AODV.

When node G receives the RREQ, the reverse pointers form a reverse path to node A. It sends a

RREP (Route Reply) packet along the reverse path to node A, as in Figure 2.4(b). The intermediate nodes and the source node set up next-hop to the destination node (node G in this example) as the node from which the RREP is received. Once the route has been established, node A can send data packets along the path.

An established route is used until the route breaks due to node failures or mobility. When an intermediate node discovers that it can no longer send a packet to its next hop, it forwards a RERR (Route Error) message back to the source so that it can restart the route discovery process.

Chapter 3

Multi-Channel MAC Protocols for Multi-Hop Wireless Networks

Multi-channel MAC protocols combine channel assignment and coordination with medium access control. Thus, the MAC protocol decides when to transmit a packet, as well as which channel it should transmit the packet on. In this chapter, we first review previous work in the area of multi-channel MAC. Then, we identify issues regarding medium access control in multi-channel environment and discuss approaches for addressing the issues. Then, we propose a multi-channel MAC protocol based on periodic channel negotiation. Finally, we evaluate the proposed protocol using simulations, and discuss the results.

3.1 Related Work

There have been efforts to exploit the benefit of using multiple channels. Dual Busy Tone Multiple Access [24] divides a common channel into two sub-channels, one data channel and one control channel. Busy tones are transmitted on a separate control channel to avoid hidden terminals, while data is transmitted on the data channel. This scheme uses only one data channel and is not intended for increasing throughput using multiple channels.

Hop Reservation Multiple Access [25] is a multi-channel protocol for networks using slow frequency hopping spread spectrum (FHSS). The hosts hop from one channel to another according to a predefined hopping pattern. When two hosts agree to exchange data by an RTS/CTS handshake, they stay on the same frequency for communication. Other hosts continue hopping, and more than one communication can take place on different frequency hops. Receiver Initiated Channel-Hopping with Dual Polling [26] takes a similar approach, but the receiver initiates the collision avoidance handshake instead of the sender. These schemes can be implemented using only one transceiver for each host, but they only apply to frequency hopping networks, and cannot be used in systems using other mechanisms such as direct sequence spread spectrum (DSSS).

Nasipuri et al. [27] propose a multi-channel CSMA protocol with "soft" channel reservation. If there are N channels, the protocol assumes that each host can listen to all N channels concurrently. A host wanting to transmit a packet searches for an idle channel and transmits on that idle channel. Among the idle channels, the one that was used for the last successful transmission is preferred. In [28] the protocol is extended to select the best channel based on signal power observed at the sender. These protocols require N transceivers for each host, which can be very expensive, especially when there are large number of channels.

Wu et al. [29] propose a protocol that assigns channels dynamically, in an on-demand style. In this protocol, called *Dynamic Channel Assignment* (DCA), they maintain one dedicated channel for control messages and other channels for data. Each host has two transceivers, so that it can listen on the control channel and the data channel simultaneously. RTS/CTS packets are exchanged on the control channel, and data packets are transmitted on the data channel. In RTS packet, the sender includes a list of preferred channels. On receiving the RTS, the receiver decides on a channel and includes the channel information in the CTS packet. Then, DATA and ACK packets are exchanged on the agreed data channel. This protocol does not need synchronization and can utilize multiple channels with little control message overhead. But it does not perform well in an environment where all channels have the same bandwidth. When the number of channels is small, one channel dedicated for control messages can be costly. For example, in the case of IEEE 802.11b, only 3 channels are available. If one channel is assigned as a control channel, it takes up 33% of the available bandwidth. On the other hand, if the number of channels is large, the control channel can become a bottleneck and prevent data channels from being fully utilized.

Jain et al. [30] propose a protocol that uses a scheme similar to [29] that has one control channel and N data channels, but selects the best channel according to the channel condition at the receiver side. The protocol achieves throughput improvements by intelligently selecting the data channel, but still has the same disadvantages as DCA.

Finally, Caccamo et al. [71] have proposed a multi-channel medium access scheme for real-time sensor network systems. In the proposed scheme, the sensor network is divided into cells, and each cell uses different channel. For intra-cell communication, a common channel is used. To communicate across cells, each cell has a designated router that exchange messages between cells. The inter-cell communications uses TDMA, in which time frames are assigned to sender-receiver pairs. The designated routers are equipped with two interfaces, so that they can access two channels simultaneously. When two routers communicate, the sender uses its own cell frequency to transmit, while the receiver uses the channel of the cell from which it expects to receive. This scheme takes a cluster-based approach in using multiple channels, and only require some nodes (designated routers) to have multiple interfaces.

3.2 Issues in Multi-channel Environment

Now we discuss the issues in multi-channel environment that are important in designing multi-channel MAC protocols. In particular, we look at channel synchronization, channel assignment, and channel scheduling.

3.2.1 Channel Synchronization

There are many MAC protocols already being used in single-channel networks, such as IEEE 802.11 DCF. So a natural question to ask is: can we directly apply the protocol to multi-channel environment? The answer is no, because in order to communicate, the sender and the receiver has to be *operating on the same channel*. (As mentioned before, we focus on the situation where each node is equipped with a single interface, so a node can only listen to one channel at a time.) When nodes A and B are on the same channel, we say that their channels are *synchronized*. Before transmitting a packet, the sender must know or ensure that the receiver is listening on the same channel. IEEE 802.11 DCF assumes that every node in the same network shares a common channel, so when a node transmits a packet, noeds in the sender's transmission range can receive the packet. However, in a multi-channel environment, nodes within reach of the sender may not receive the packet, if they are listening on a different channel. Thus, DCF cannot be directly applied to a multi-channel network.

To make a basis for discussion, suppose we extend IEEE 802.11 DCF for multi-channel networks as follows. Initially, each node chooses a random channel among N channels (N is the total number of available channels), and this channel information is known to every other node in the network. In Figure 3.1, nodes A, B, C are assigned channels 1, 2, and 3, respectively. When node A wants to send a packet to node B, it switches to channel 2 and contends for the channel as in IEEE 802.11 DCF. After node A sends the packet to B, it returns to channel 1, where it can receive packets from other nodes.

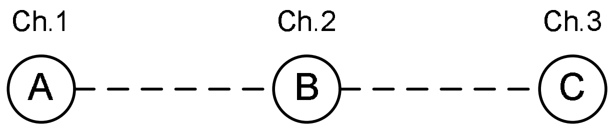


Figure 3.1: An example multi-channel network scenario with 3 nodes.

This simple multi-channel MAC protocol can suffer significant performance degradation due to the *Channel Synchronization Problem*. In Figure 3.1, when node A is sending a packet to B on channel 2, node B might be on channel 3, waiting to send a packet to C. Since node A does not know which channel B is on, it can go ahead and send the packet, which cannot be received at B. Then, node A cannot distinguish whether this packet loss is due to collision, or the fact that B is not on channel 2. Following the retransmission procedure in IEEE 802.11 DCF, node A retransmits the packet, thinking that the previous failure was due to collision. If node A fails to send a packet to B several times, it concludes that the link between A and B has been broken, drops the packet and notifies upper layer protocols that the link is no more valid. However, the packet loss was due to the fact that node B was on another channel, not because of link failure.

To avoid this undesirable series of events, node A must be sure (at least with high confidence) that the channel between A and B is synchronized, at the point of time when A sends a packet. There are three possible approaches to ensure channel synchronization.

- Approach 1: Each node switches channels according to a patterend schedule which is known locally to its neighbors, or globally known to all other nodes. The channel schedule can be determined prior to deployment, or decided on-the-fly and advertised to neighbors using messages. For example, suppose node A decides to switch between channel 2 and channel 3 every 100 ms, and this schedule is known to its neighbor, node B. Then, without further message exchange, node B will know when and on what channel it can send packets to A, until A changes its schedule. Whenever a node changes its channel schedule, the new schedule must be advertised to its neighbors. This approach requires clocks to be synchronized. Since nodes in the network often have different clock drifts, a clock synchronization service may need to be run independently for this approach to work.

- Approach 2: Nodes may stay on any channel of their choice, but they have a prior agreement that all nodes must switch to a common channel periodically, and stay on the common channel for a certain duration of time. With this agreement, the sender knows when and on what channel it can schedule its packet transmission for the receiver. During the period of time where all nodes are on the common channel, nodes can negotiate with each other so that communicating pair of nodes may stay on the same channel for a longer duration of time. This approach also requires clock synchronization.

- Approach 3: Each node switches channels according to its own schedule which may not have any pattern, and it informs its neighbors whenever it switches channels. The overhead of this approach is most expensive, because it needs to send messages every time it switches channels. So, this approach can be effective when a node stays on a channel for a long time. For example, when a route is established, the route can be assigned a channel which does not change for a long period of time.

3.2.2 Channel Assignment

In multi-channel networks, channel assignment strategy is critical in achieving high channel utilization. Ideally, it is best to have the traffic equally divided among channels. However, it is difficult to achieve perfect channel load balancing due to dynamic channel conditions and traffic patterns. Still, a multi-channel protocol should allocate channels properly so that high channel utilization is achieved.

Channels can be assigned in two ways: proactive and reactive. A proactive channel assignment scheme allocates channels based on traffic distribution, so that load is balanced among channels. The limitation of this scheme is that it requires a central channel coordinator which knows the entire topology of the network. Also, the coordinator needs to know the traffic pattern, so that it can properly assign channels. Thus, this scheme will not work well in a multi-hop network, where a central coordinator does not exist, and every node has limited information on network topology and traffic. In a reactive channel assignment scheme, the “best” channel is chosen at the time when traffic is to be sent, according to the information obtained on current channel conditions. Since each node makes decisions independently, no central coordinator is required, and no prior knowledge on traffic is required. So reactive channel assignment scheme is more feasible for multi-hop networks.

The trade-off in channel assignment lies between the cost of gathering information on channel conditions and the effectiveness of channel selection. In one extreme case, a pair of nodes may not have any information on channel conditions (e.g. what channels their neighbors are using). Then they will choose a random channel, which may lead to imbalance in channel load. On the other extreme, a node may obtain everything it needs to decide which channel has the minimum use. Then it can make the “best” choice, but the nodes have to exchange messages with other nodes to obtain the channel information. The accuracy increases at the cost of exchanging messages frequently.

3.2.3 Channel Scheduling

A node may need to send packets to multiple neighbors on different channels. Figure 3.2 illustrates an example case, where node A needs to switch between channels 2 and 3 to send packets. In this case, node A must schedule time for staying on each channel. Switching channels on per-packet basis is undesirable because of the channel switching delay. On the other hand, if a node stays on one channel for too long, the packets that needs to be delivered on other channels must wait longer, which increases the packet delay.

There are several approaches for scheduling channels, which can affect the throughput, average packet delay and packet delay variance.

- fixed channel scheduling: The most simple method is to schedule channels with fixed period in a round robin manner. In Figure 3.2, node A can stay on channel 2 for 100 ms, and then stay on channel 3 for 100 ms. While staying on a channel, it can go on to the next channel if there is no packet pending on the current channel. This method can make the traffic bursty,

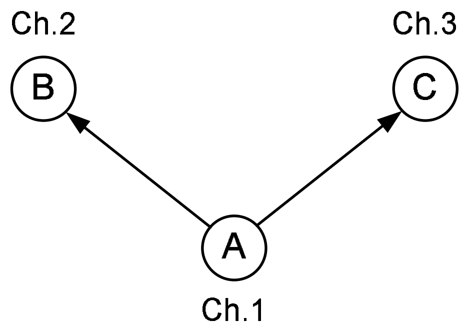


Figure 3.2: An example case where a node needs to send packets to multiple destinations on different channels.

which may not be desirable for flows that need low jitter. Also, if the load is different among channels, this approach may lead to inefficient channel utilization.

- scheduling based on pending packets: The node can schedule time based on the amount of pending packets in the queue. A high-rate flow can benefit from this method, but a low-rate flow may suffer a very high delay, because the amount of channel occupation decreases.

- first-come first-serve: Regardless of the channel, a node can switch to the channel where the first packet in the queue needs to be sent. In the worst case, the node may need to switch channels for every packet, and the overhead of channel switching delay may significantly affect the performance.

- delay limit scheduling: To reduce the impact of channel switching delay, this method tolerates a packet delay up to a certain bound. So a node can stay on a particular channel, until a packet to be sent on a different channel has waited more than the threshold. Then it switches to the channel and stays there until it needs to switch because of the delay limit. This method may give larger amount of channel occupation to high-rate flows, while protecting low-rate flows from waiting too long in the queue.

3.3 MMAC: A Multi-Channel MAC Protocol with Periodic Channel Negotiation [1]

Based on the discussion from the previous sections, we propose a multi-channel MAC protocol for wireless ad hoc networks. The protocol called MMAC (Multi-channel MAC Protocol) uses periodic channel negotiation to assign channels and address the channel synchronization issue. MMAC divides time into beacon intervals, where every node in the network is synchronized. At the start of each beacon interval, every node switches its channel to a common channel. So channel synchronization is done using globally known information. At the start of each beacon interval, nodes exchange messages to select data channel for the particular beacon interval. The "best" channel is selected according to the traffic information obtained from overhearing the channel negotiation messages. After channel negotiation, each node switches to its selected channel and stay on the channel until the end of the beacon interval. In this section, we describe MMAC in detail, and provide some experimental results to show that MMAC can utilize multiple channels and achieve high channel utilization.

3.3.1 Background: IEEE 802.11 Power Saving Mechanism (PSM)

Here we describe IEEE 802.11 PSM to explain how ATIM windows are used. A node can save energy by going into *doze* mode. In doze mode, a node consumes much less energy compared to normal mode, but cannot send or receive packets. It is desirable for a node to enter the doze mode only when there is no need for exchanging data. In IEEE 802.11 PSM, this power management is done based on *Ad hoc Traffic Indication Messages* (ATIM). Time is divided into beacon intervals, and every node in the network is synchronized by periodic beacon transmissions. So every node will start and finish each beacon interval at about the same time.

Figure 3.3 illustrates the process of IEEE 802.11 PSM. At the start of each beacon interval, there exists an interval called the ATIM window, where every node should be in the awake state. If node A has buffered packets for B, it sends an ATIM packet to B during this interval. If B receives this message, it replies back by sending an ATIM-ACK to A. Both A and B will then stay awake for that entire beacon interval. If a node has not sent or received any ATIM packets during the

ATIM window (e.g., node C in Figure 3.3), it enters doze mode until the next beacon time.

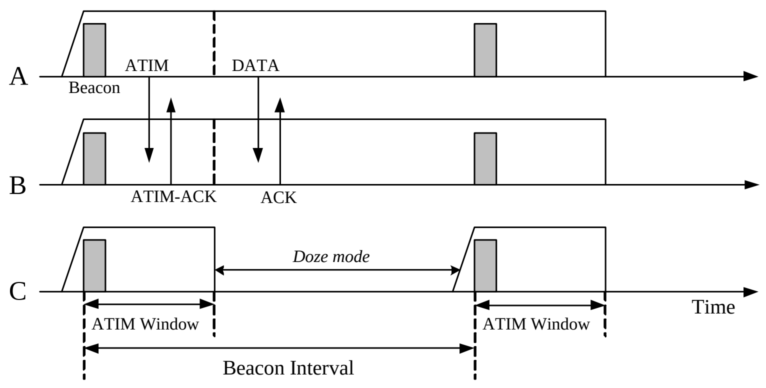


Figure 3.3: Operation of IEEE 802.11 PSM.

3.3.2 Proposed Multi-Channel MAC (MMAC) Protocol

In this section, we present our proposed scheme. Before describing the protocol in detail, we first summarize our assumptions.

- N channels are available for use and all channels have the same bandwidth. None of the channels overlap, so the packets transmitted on different channels do not interfere with each other. Hosts have prior knowledge of how many channels are available.
- Each host is equipped with a single half-duplex transceiver. So a host can either transmit or listen, but cannot do both simultaneously. Also, a host can listen or transmit on only one channel at a time. So when listening to one channel, it cannot carrier sense on other channels. Unlike our scheme, many other multi-channel MAC protocols require each host to have multiple transceivers [27, 29, 30].
- The transceiver is capable of switching its channel dynamically. The time elapsed for switching the channel is assumed to be $224\mu s$ [5].
- Nodes are synchronized, so that all nodes begin their beacon interval at the same time. Clock synchronization can be achieved using either out-of-band solutions such as GPS, [31], or in-band solutions. If an out-of-band solution can be used, no additional overhead is imposed on the channels used by our protocol. However, if an in-band solution is used, we need to consider

the overhead of synchronization. To model this overhead, we implement beaconing mechanism similar to IEEE 802.11 timing synchronization function (TSF) [5] in our simulations (although the clocks are perfectly synchronized in the simulations). The issue of clock synchronization is discussed further in Chapter 4.

Now we describe our proposed scheme in detail. From now on, our protocol will be referred as *Multi-channel MAC* (MMAC).

Preferable Channel List (PCL)

Each node maintains a data structure called the *Preferable Channel List* (PCL), that indicates which channel is preferable to use for the node. PCL records the usage of channels inside the transmission range of the node. Based on this information, the channels are categorized into three states.

- *High preference* (HIGH): This channel has already been selected by the node for use in the current beacon interval. If a channel is in this state, this channel must be selected. For each beacon interval, at most one channel can be in this state at each node.
- *Medium preference* (MID): This channel has not yet been taken for use in the transmission range of the host. If there is no HIGH state channels, a channel in this state will be preferred. (The selected channel enters HIGH state.)
- *Low Preference* (LOW): This channel is already taken by at least one of the node's immediate neighbors. To balance the channel load as much as possible, there is a counter for each channel in the PCL to record how many source-destination pairs plan to use the channel for the current interval. If all channels are in LOW state, a node selects the channel with the smallest count. (The selected channel enters HIGH state.)

The channel states are changed in the following way:

- All the channels in the PCL are reset to MID state when the node is powered up, and at the start of each beacon interval.

- If the source and destination nodes agree upon a channel, they both record the channel to be in HIGH state.

- If a node overhears an ATIM-ACK or ATIM-RES packet (explained later), it changes the state of the channel specified in the packet to be LOW, if it was previously in the MID state. When the state of a channel changes from MID to LOW, the associated counter is set to one. If the channel was previously in HIGH state, it stays in the HIGH state. If the channel was already in the LOW state, the counter for the channel is incremented by one.

Channel Negotiation during ATIM Window

In MMAC, periodically transmitted beacons divide time into beacon intervals. A small window called the *ATIM window* is placed at the start of each beacon interval (we use the term "ATIM" as in IEEE 802.11 PSM, although it is used for a different purpose in our protocol). The nodes that have packets to transmit negotiate channels with the destination nodes during this window. In the ATIM window, every node must listen to the *default channel*. The default channel is one of the multiple channels, which is predefined so that every node knows which channel is the default channel. During the ATIM window, all nodes listen on the default channel, and beacons and ATIM packets are transmitted on this channel. Note that outside the ATIM window, the default channel is used for sending data, similar to other channels.

If node S has buffered packets destined for D, it will notify D by sending an ATIM packet. S includes its preferable channel list (PCL) in the ATIM packet. D, upon receiving the ATIM packet, selects one channel based on the sender's PCL and its own PCL. As explained in the next section, the receiver's PCL has higher priority in selecting the channel. After D selects a channel, it includes the channel information in the ATIM-ACK packet and sends it to S. When S receives the ATIM-ACK packet, it sees if it can also select the channel specified in the ATIM-ACK. S can select the specified channel only except when S has already selected another channel (according to rules for selecting the channel, explained in the subsequent section). If S selects the channel specified in the ATIM-ACK, S sends an ATIM-RES packet to the D, with S's selected channel specified in the packet. (If S cannot select the channel specified in the ATIM-ACK, S does not send an ATIM-RES to D). The ATIM-RES (ATIM-Reservation) is a new type of packet used in our scheme, which is not in IEEE 802.11 PSM. The ATIM-RES packet notifies the nodes in the vicinity of S which channel S is going to use, so that the neighboring nodes can use this information to update their PCL. Similarly, the ATIM-ACK packet notifies the nodes in the vicinity of D. After the ATIM window, S and D will switch to the selected channel and start communicating by exchanging RTS/CTS.

When multiple nodes start sending ATIM packets at the beginning of a beacon interval, ATIM packets will collide with each other. To avoid such collisions, nodes apply the binary exponential backoff mechanism (explained in Chapter 2) when transmitting ATIM packets. Also, similar to RTS and CTS packets, ATIM and ATIM-ACK packets also include NAV information to avoid hidden terminal problems in a multi-hop network.

Note that the receiver can always select a channel for use. Even if all the channels are selected for use in the receiver's transmission range, the receiver can select one of the channels. This is possible because the sender and receiver still exchange RTS/CTS before sending DATA packet, after the ATIM window. If two source-destination pairs that are closely placed choose the same channel, they will have to contend with each other just as in original IEEE 802.11.

Power saving is not the main goal of our protocol, but a node may save power by going into *doze* mode, if it has not transmitted or received ATIM packets during the ATIM window. The possibility of integration with IEEE 802.11 PSM is one advantage of our protocol. However, in our simulations, nodes do not go into doze mode.

Rules for Selecting the Channel

When a node receives an ATIM packet, it selects a channel and notifies the sender by including the channel information in the ATIM-ACK packet. The receiver tries to select the "best" channel based on information included in the sender's PCL (preferable channel list) and its own PCL. By the best channel we mean the channel with the least scheduled traffic, as elaborated below. This selection algorithm attempts to balance the channel load as much as possible, so that bandwidth waste caused by contention and backoff is reduced. For this reason, we count the number of source-destination pairs that have selected the channel by overhearing ATIM-ACK and ATIM-RES packets and select the one with the lowest count. This scheme assumes that every source-destination pair will deliver the same amount of traffic in a beacon interval, which may not be true. A better approach may be to count the number of packets scheduled to be transmitted on the channel in the beacon interval. To do this, the source needs to include the number of pending packets in the ATIM packet. We take the former approach here.

Now we describe the channel selection algorithm in detail. Suppose that node A has packets for B and thus sends an ATIM packet to B during the ATIM window, with A's PCL included in the packet. On receiving the ATIM request from A, B decides which channel to use during the beacon interval, based on its PCL and A's PCL. The selection procedure used by B is described as follows.

1. If there is a HIGH state channel in B's PCL, this channel is selected.

2. Else if there is a HIGH state channel in A's PCL, this channel is selected.

3. Else if there is a channel which is in the MID state at both A and B, it is selected. If there are multiple channels in this state, one is selected arbitrarily.

4. Else if there is a channel which is in the MID state at only one node, A or B, it is selected. If there are multiple of them, one is selected arbitrarily.

5. If all of the channels are in the LOW state, add the counters of the sender's PCL and the receiver's PCL. The channel with the least count is selected. Ties are broken arbitrarily.

After selecting the channel, B sends an ATIM-ACK packet to A, specifying the channel it has chosen. When A receives the ATIM-ACK packet, A will see if it can also select the channel specified in the ATIM-ACK packet. If it can, it will send an ATIM-RES packet to B, with A's selected channel specified in the packet. If A cannot select the channel which B has chosen, it does not send an ATIM-RES packet to B.

The process of channel negotiation and data exchange in MMAC is illustrated in Figure 3.4. During the ATIM window, A sends ATIM to B and B replies with ATIM-ACK indicating to use channel 1. This ATIM-ACK is overheard by C, so channel 1 will be in LOW state in C's PCL. When D sends ATIM to C, C selects channel 2. After the ATIM window, the two communications (between A and B, and C and D) can take place simultaneously.

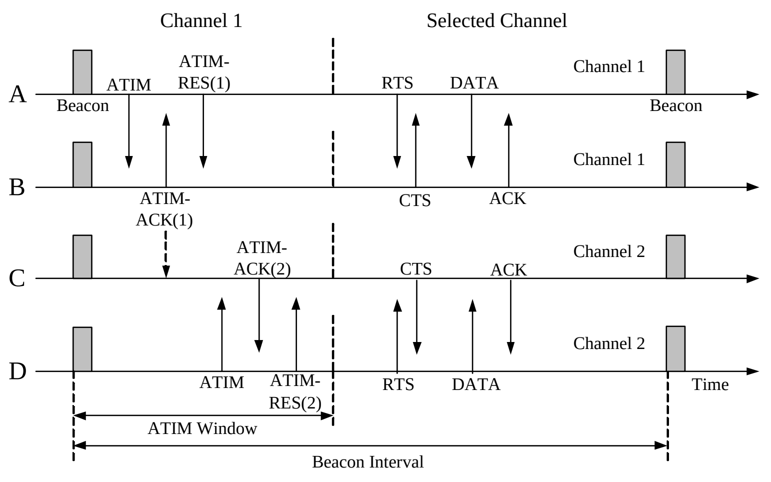


Figure 3.4: Process of channel negotiation and data exchange in MMAC.

3.4 Performance Evaluation

In this section we evaluate the performance of our protocol by simulation. We compare our scheme with IEEE 802.11, and the Dynamic Channel Assignment (DCA) protocol [29] (DCA was explained in Section 3.1). Recall that the DCA protocol uses a separate channel for exchanging control messages and uses other channels for data. This approach is also taken by [30, 32]. We used two metrics to evaluate the performance of our protocol, aggregate network throughput and average packet delay. The aggregate network throughput is the sum of throughput over all flows in the network. The average packet delay is the duration between the time when the link layer of the sender receives a packet to send, and the time the packet reaches the destination. So the packet delay is the sum of delays for queueing, backoff, channel negotiation and transmission delay. We ignore lost packets in the average delay measurement.

3.4.1 Simulation Model

For simulations, we have used ns-2 [33] with CMU wireless extensions [34]. Simulations are performed in two network scenarios, wireless LAN and multi-hop networks. The bit rate for each channel is 2Mbps. The transmission range of each node is approximately $250m$ and the beacon interval is set to $100ms$. Each source node generates and transmits constant-bit rate (CBR) traffic. Each simulation was performed for a duration of 40 seconds. Each data point in the result graphs is an average of 30 runs.

Unless otherwise specified, we assume 3 channels. Also we assume packet size is 512 bytes, and ATIM windows are 20 ms unless specified otherwise. The parameters we vary are: number of nodes in the network, the packet arrival rate of CBR traffic, ATIM window size, and number of channels.

Wireless LAN

In the simulated wireless LAN, all nodes are within each other's transmission range. So every source node can reach its destination in a single hop. The number of nodes we used are 6, 30, and 64. For each scenario, half of the nodes are sources and the other half are destinations. So a source has at most one destination. The impact of a source having multiple destinations or a destination having multiple sources is not studied in this scenario, but it is studied in the multi-hop network scenario.

First, we examine the throughput and packet delay varying the network load. We use the packet arrival rate of CBR flows to vary the network load. After that, we study the impact of ATIM window size and number of available channels on the throughput.

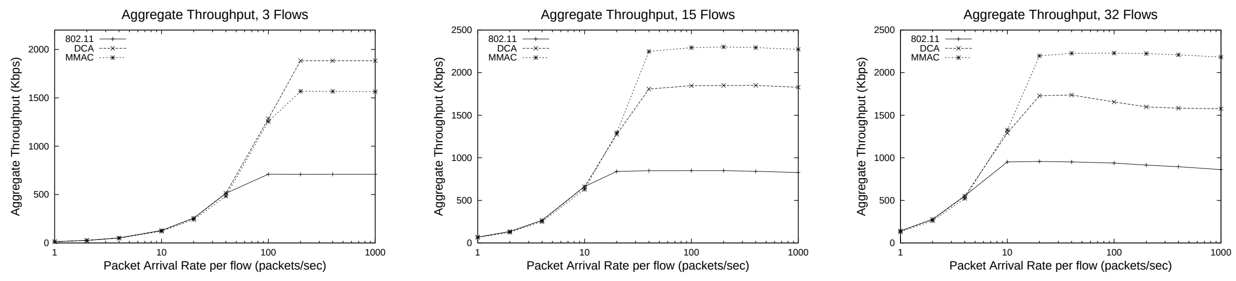
Multi-hop network

For a multi-hop network, 100 nodes are randomly placed in a $500m \times 500m$ area. 40 nodes are randomly chosen to be sources, and 40 nodes are chosen to be destinations. A node may be the source for multiple destinations and a node may be the destination for multiple sources. In a multi-hop network, we study the situation where different traffic loads are present in different regions, which is not done in the wireless LAN scenario.

3.4.2 Simulation Results

Simulation results are presented in this section. In the graphs, the curves labeled as "802.11" refer to original IEEE 802.11 single channel MAC, the curves labeled as "DCA" indicate the DCA protocol from [29], and the curves labeled as "MMAC" indicate our proposed scheme.

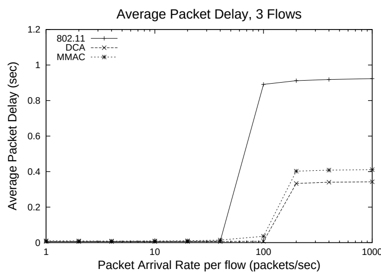
First we present results from simulations performed for a wireless LAN. Figure 3.5 shows the aggregate throughput of different protocols as the network load increases. The network sizes are 6, 30, and 64 nodes in Figure 3.5(a), (b), and (c) respectively. When network load is low, all protocols perform similarly. As network load draws near saturation, MMAC performs significantly better than IEEE 802.11, and also does better than DCA. Since there are 3 channels, DCA uses 1 channel for control packets and other 2 channels for data. By using this separate control channel, DCA achieves almost twice the throughput of IEEE 802.11. But as the number of channels increases, the throughput improvement of DCA for the added channel becomes less, because of bottleneck on control channel, as we will see later. MMAC uses all 3 channels for data exchange, but cannot achieve 3 times as much throughput compared to IEEE 802.11 because of its overhead for channel negotiation. The overheads in MMAC are periodic beacon transmissions and ATIM packets. As the graphs show, MMAC can perform 20%-30% better than DCA. The throughput improvement of MMAC over DCA may not be dramatic, but it is important that MMAC achieves this throughput using only a single transceiver per node.



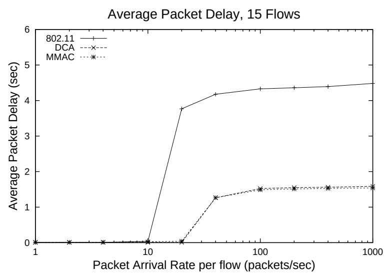
(a) 6 nodes (b) 30 nodes (c) 64 nodes

Figure 3.5: Aggregate Throughput vs. Packet Arrival Rate in a wireless LAN.

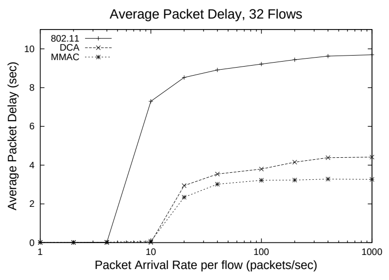
Figure 3.6 shows the average packet delay of the protocols as the network load increases. The difference between IEEE 802.11 and other protocols in delay is due to the fact that with only one channel, a packet has to wait longer to use the channel when the network load is high. When comparing DCA and MMAC, MMAC shows higher delay in the network scenario with 6 nodes. Then the delay of the two protocols becomes similar with 30 nodes, and MMAC outperforms DCA in 64-node scenario. When the number of nodes are small, MMAC shows higher delay because packets have to wait during the channel negotiation phase (ATIM window). But when the number of nodes becomes large, DCA suffers from high contention at the control channel which results in high packet delay. MMAC does not have this problem, because it does not maintain a separate channel for control messages.



(a) 6 nodes



(b) 30 nodes

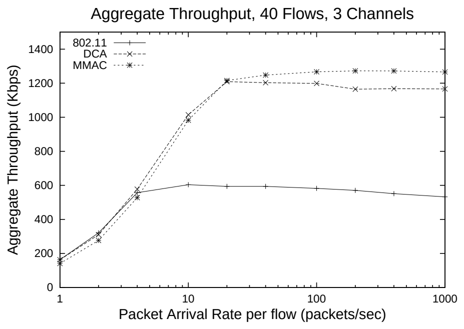


(c) 64 nodes

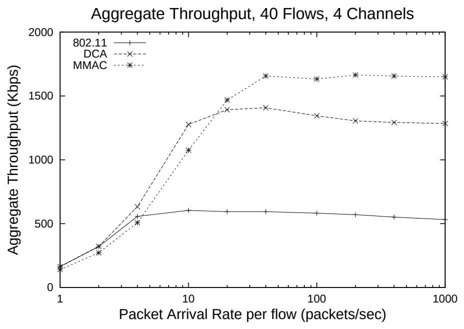
Figure 3.6: Average Packet Delay vs. Packet Arrival Rate in a wireless LAN.

Now we look at results from a multi-hop network. As stated in the previous section, in our multi-hop network simulations, a node can be a source for multiple destinations, or it can be a destination for multiple sources. Figure 3.7 shows the aggregate throughput of different protocols as the network load increases. Three and four channels are used in Figure 3.7(a) and 3.7(b) respectively. In 3.7(a), MMAC performs better than DCA, but the difference is smaller than in the wireless LAN case. This is due to the following reasons. First, in a multi-hop network, all 3 channels may not be fully utilized throughout the entire area. In the region where the network can benefit from having the third channel, MMAC does better than DCA. But in the region where only two channels are needed, DCA does better than MMAC because it can utilize 2 data channels without ATIM window overhead. Second, in MMAC, if a node has flows to two different destinations, each destination may choose a different channel and one flow may have to wait for an entire beacon interval to negotiate the channel again. Also, if a node is a destination for two flows from other sources, these two flows must be transmitted on the same channel, reducing the benefit of having multiple channels. As the network load becomes very high, throughput of DCA drops faster than MMAC. This is because a single control channel is shared by every node in DCA. When the network load is very high, the collision rate of control packets increases, degrading the throughput. We call this *control channel saturation*.

The impact of control channel saturation is also shown in Figure 3.7(b). MMAC gains significant benefit from not having a dedicated control channel. In [29], it is shown that the maximum number of channels that can be fully utilized using DCA is $L_d/3L_c$, given that L_d is the data packet size and L_c is the control packet size. But even when the number of channels is less than $L_d/3L_c$, the



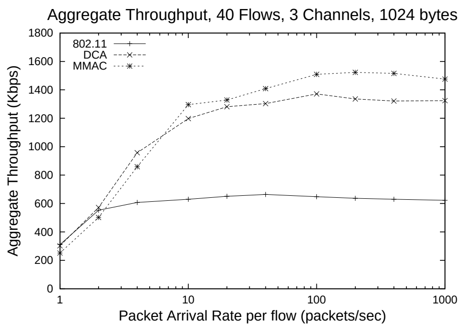
(a) 3 channels



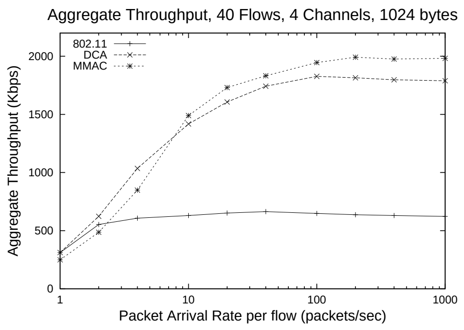
(b) 4 channels

Figure 3.7: Aggregate Throughput vs. Packet Arrival Rate in a multi-hop network. Packet size is 512 bytes.

throughput suffers from high contention among the control packets, especially when the network load is high. The impact of control channel saturation can be reduced if larger packets are used, because less control message are needed to transmit the same amount of data.The results using larger packets are shown in Figure 3.8(a) and 3.8(b). As the results show, MMAC only does slightly better than DCA, both with three and four channels. However, MMAC achieves this improvement with simpler hardware than DCA.



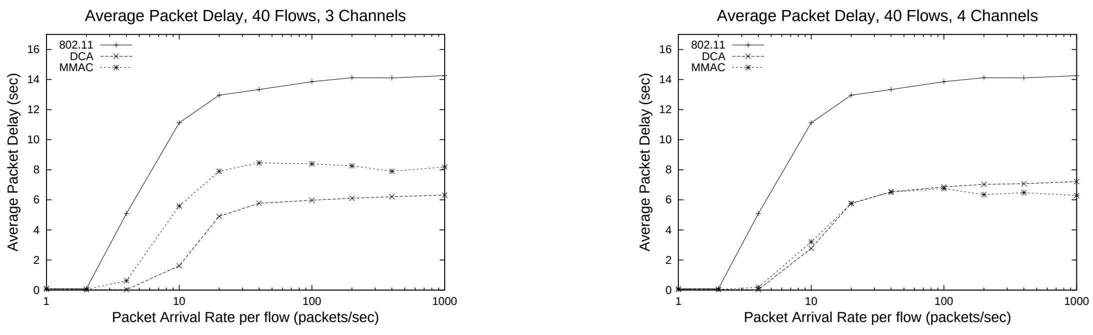
(a) 3 channels



(b) 4 channels

Figure 3.8: Aggregate Throughput vs. Packet Arrival Rate in a multi-hop network. Packet size is 1024 bytes.

Figure 3.9(a) and 3.9(b) shows the average packet delay of the protocols as the network load increases. Packet size is again 512 bytes in these graphs. With three channels, MMAC shows higher delay than DCA, even though MMAC achieves higher throughput. This is due to the same reasons explained in the wireless LAN scenario. However, when four channels are available, MMAC shows lower delay than DCA. We can see that the average packet delay of DCA is almost the same with three and four channels. This is because DCA does not benefit from having one more channel because of control channel saturation, as previously mentioned. But MMAC benefits from having the fourth channel and the average delay becomes lower. ATIM window overhead in MMAC does not increase with the number of channels, as long as the ATIM window is long enough to exchange all the ATIM messages necessary.



(a) 3 channels (b) 4 channels

Figure 3.9: Average Packet Delay vs. Packet Arrival Rate in a multi-hop network.

We have fixed the ATIM window size to $20ms$ until now. But this may be undesirable due to the following reason. When there are small number of flows in the network, using 20% of each beacon interval for exchanging ATIM messages is wasteful. Much of the time the channel will be left as idle, because data packets are not allowed to be transmitted in this interval. On the other hand, if there are very large number of flows in the network, a longer ATIM window would be needed to exchange all the ATIM messages between nodes to negotiate channels. Thus the ATIM window size affects the throughput of MMAC protocol[1]. To study this impact, aggregate throughput is measured using different ATIM window sizes and shown in Figure 3.10. The three curves show results for 256, 512 and 1024 bytes of packet size. For this network scenario, an ATIM window size of around 15-20ms is shown to be the best for throughput. When the ATIM window size is too small, not all nodes can exchange ATIM messages and negotiate channels during this interval. Nodes that have not successfully exchanged ATIM messages stay on the default channel. So the multiple channels cannot be fully utilized, resulting in degraded throughput. If the ATIM window size is too large, the throughput decreases because no benefit is obtained with increased

[1]A similar observation about the impact of ATIM window on IEEE 802.11 PSM is reported in [35, 36].

overhead from having a longer ATIM window. The optimal ATIM window size depends mainly on the number of flows in the beacon interval, because an ATIM packet exchange is required for each flow. Since the number of flows is often dynamically changed, it is desirable to also make the ATIM window size adapt to the network situation. Changing ATIM window size dynamically to achieve maximum throughput is not addressed in this dissertation. [2]

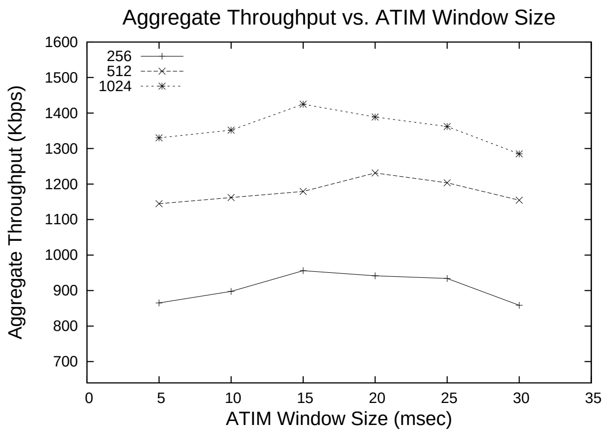
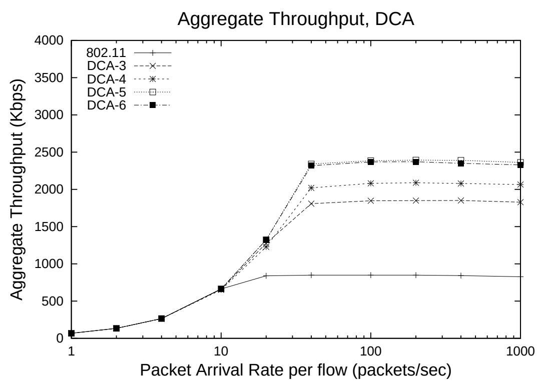


Figure 3.10: Aggregate Throughput vs. ATIM Window Size in a multi-hop network.

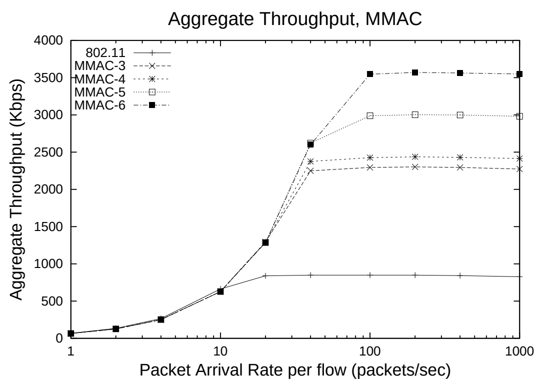
Finally, we measured the throughput of different protocols varying number of channels. This simulation was done for a wireless LAN with 30 nodes. We used 512 bytes for packet size, and the number of channels vary from 3 to 6. The results are shown in Figure 3.11. In the graphs, "MMAC-3" indicates MMAC protocol with 3 channels, and "DCA-4" refers to DCA protocol with 4 channels. The throughput of IEEE 802.11 is also shown in the graphs. Because of control channel saturation, DCA does not benefit from having additional channels when the number of channels becomes larger. MMAC does better than DCA with the same number of channels, especially when the network load is high.

DCA and MMAC have their own ways to avoid the hidden terminal problem and access multiple channels dynamically. DCA uses one separate channel to exchange control packets, whereas MMAC uses ATIM windows to negotiate channels. In DCA, the bandwidth of the control channel has a major impact on the performance. In MMAC, the ATIM window size takes the role. Under our simulation model, MMAC, which uses simpler hardware than DCA, performed better than DCA.

[2]Changing ATIM window size dynamically to improve throughput in IEEE 802.11 PSM is studied in [36].



(a) DCA



(b) MMAC

Figure 3.11: Aggregate Throughput vs. Packet Arrival Rate in a wireless LAN (30 nodes).

In our simulations, we assume that all channels have the same bandwidth, as in IEEE 802.11 specification. However, if we can control the bandwidth of data and control channels, performance of the DCA protocol can be improved further by tuning the control channel bandwidth properly. The optimal control channel bandwidth in DCA depends on the traffic load, just as the optimal ATIM window size in MMAC. Even though as of now we do not have a scheme that dynamically controls ATIM window size based on traffic load, it is much easier to adjust ATIM window size dynamically than to adjust bandwidth of the channels. Because of this reason, MMAC has higher flexibility than DCA.

3.5 Discussion

In this section, we discuss some issues related to our scheme, and possible ways to improve it.

As stated earlier, the MMAC protocol requires all clocks in the network to be synchronized, so that all nodes start a beacon interval at the same time. Clock synchronization can be achieved using either out-of-band (such as GPS) or in-band solutions. If the nodes are capable of using an out-of-band solution for synchronization, no additional overhead is imposed on channels used by our protocol. However, if an in-band solution is used, it imposes an additional overhead which might affect the performance of the protocol. To model the overhead, we have implemented the beaconing mechanism similar to IEEE 802.11 TSF, which works as follows. At the start of a beacon interval, each node waits for a random delay and transmits a beacon. If a node receives a beacon before transmitting its own beacon, it suppresses and does not transmit its beacon. Since the beacons model the overhead of synchronization, if an out-of-band solution can be used, our protocol will perform better than what our simulations show, since the overhead of beacons will not be necessary.

The beaconing mechanism we use in our simulations is meant to model the potential overhead of synchronization. IEEE 802.11 TSF is designed for wireless LANs, where all nodes are within transmission range of each other. IEEE 802.11 TSF can be applied to multi-hop networks, but in some cases it may fail to synchronize a multi-hop network, because of the following problem.

Consider the scenario in Figure 3.12. At the start of a beacon interval, each node chooses a random delay and transmits a beacon. It might happen that node A always transmits a beacon before B, and node D always transmits a beacon before C. Then the clocks of (A, B) and (C, D) may drift away, because they never exchange beacons. In the next chapter, we develop a clock synchronization protocol that can work with MMAC.

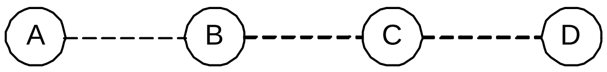


Figure 3.12: A chain topology of 4 nodes.

In MMAC, nodes switch to the common channel at the beginning of each beacon interval. However, if a node starts transmitting a data packet near the end of a beacon interval, the time when the node switches its channel may be pushed back. In this case, the node might miss the ATIM packets sent by other nodes. To prevent this, nodes refrain from transmitting a packet if the time left for the current beacon interval is less than the transmission time of the packet.

When a node is sending packets to two different destinations, these two destination nodes may select a different channel. For example, suppose that we have nodes A, B and C in the network, as in Figure 3.13. Node A has some packets destined for B and others destined for C. During channel negotiation, node B selects channel 1 and node C selects channel 2. If A selects channel 1, it can only transmit packets destined for B, and all the packets destined for C must wait until next beacon interval to negotiate the channel again. This behavior of MMAC protocol raises several issues. First, to avoid *head of line blocking* problem, the packets that cannot be transmitted because of channel mismatch must be kept in a separate buffer, and restored to the queue at the end of the beacon interval. This complicates the queue management. Also, it is possible that the same channels are selected by each node in the subsequent beacon intervals, delaying the flow from A to C. In our scheme, if node A has to send ATIM packets to B and C, A chooses randomly which one to send the packet to first. This randomness should prevent complete starvation, although there can be short-term unfairness among the flows. Instead of randomly choosing among the destinations, node A can send an ATIM packet first to the destination which is the target node of the first packet in its queue. This modification will improve the fairness of the protocol.

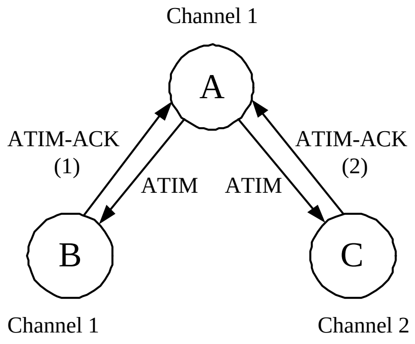


Figure 3.13: An example network scenario. Assume there are other nodes in the vicinity of these three nodes, that affect the PCL of these three nodes. Node A has packets for B, and also packets for C. A exchanges ATIM messages with B first, and both select channel 1. After that, A sends an ATIM packet to C, and C selects channel 2. Since A will stay in channel 1 for the beacon interval, packets for C must be deferred until the next beacon interval.

In addition to the problems stated above, this situation might also have impact on the throughput. Suppose that A had only a few packets for B. Then after sending all the scheduled packets, A becomes idle for the rest of the beacon interval. But A cannot send packets to C, even though A has received C's ATIM-ACK during ATIM window and knows which channel C will be listening on. To avoid waste of bandwidth, we can extend MMAC to allow nodes to switch channels inside the beacon interval. A node may switch channels according to the following rules.

- If node A finishes sending packets on its selected channel, and does not know of any node that is planning to send packets to A, A may switch its channel. If A has received any ATIM packet during the ATIM window, A must stay on the selected channel for the entire beacon interval.

- After switching to another channel, node A must wait for some time to gather information on the condition of the new channel before transmitting a packet. The amount of time it has to wait is $SIFS + t_{MTU} + p$, where t_{MTU} is the transmission delay for maximum transmission unit (MTU), and p is the propagation delay for one-hop distance. This delay is required to avoid collision, because A does not have NAV (Network Allocation Vector) information for the new channel at the time it switches channels.

In our example, node A can switch to channel 2 after sending all of its scheduled packets to B, because it has not received any ATIM packets during the ATIM window. Node C has to stay in channel 2 for the entire beacon interval, because it received an ATIM packet from A. So the communication between A and C can take place in channel 2, for the rest of the beacon interval. This extended scheme can reduce wasted bandwidth and thus increase throughput.

Another issue in MMAC is channel load balancing. In the original MMAC, a node counts the usage of a channel based on the ATIM-ACK or ATIM-RES packets it overhears, and selects the channel with the least count to balance the channel load. It means that the node is counting the number of *source-destination pairs*. The assumption here is that every flow has the same amount of traffic ready to be transmitted in the beacon interval, which may not be true. Different flows may have different number of packets pending to be sent. So it may be better to count the number of *pending packets* for each channel rather than number of source-destination pairs. To do this, the source node counts the number of pending packets for the destination and include the value in the ATIM packet. This number is echoed in the ATIM-ACK and ATIM-RES packet, so that the nodes in the vicinity of the source or destination can obtain the information. When selecting a channel, the node selects a channel with the least number of packets scheduled on the channel. This selection mechanism will achieve a better load balancing than the original scheme.

3.6 Summary

In this chapter, we have looked at issues in designing MAC protocols for multi-channel networks. We have presented MMAC, a multi-channel MAC protocol that uses periodic channel negotiations to utilize multiple channels in multi-hop wireless networks. The proposed scheme requires only one transceiver for each host, while other multi-channel MAC protocols require multiple transceivers for each host [29, 30, 37]. In order to avoid the multi-channel hidden terminal problem, we require nodes to be synchronized, so that every node starts each beacon interval at about the same time. At the start of each beacon interval, every node listens on a common channel to negotiate channels in the ATIM window. After the ATIM window, nodes switch to their agreed channel and exchange messages on that channel for the rest of the beacon interval.

Simulation results show that MMAC successfully exploits multiple channels to improve total network throughput over IEEE 802.11 single-channel. The performance of MMAC and DCA depends on the network situation, but as the simulation results show, MMAC performs better or at least comparable to DCA in many cases. It is important to note that MMAC achieves this performance using simpler hardware than DCA. Since MMAC only requires one transceiver per host, it can be implemented with hardware complexity comparable to IEEE 802.11. Also, power saving mechanism used in IEEE 802.11 can easily be integrated with MMAC for energy efficiency, without further overhead.

Chapter 4

A Clock Synchronization Service to Support Synchronous Operations in Multi-Hop Networks

The MMAC protocol proposed in Chapter 3 requires nodes to have their clocks synchronized, since all nodes have to switch to a common channel at the same time. IEEE 802.11 has a protocol called Timing Synchronization Function (TSF), which is explained in detail in Section 4.3. TSF uses timestamped beacon messages transmitted at the start of each beacon period to synchronize clocks among nodes. The protocol is designed for wireless LANs, where there is a direct link between any pair of nodes. Huang and Lai [38] point out that TSF is not scalable, meaning that as the number of nodes increase, the possibility that the nodes go out of synchronization becomes large enough to significantly impact protocols that rely on synchronized clocks. They propose a simple modification to TSF to improve the scalability, as explained in Section 4.2.

For a multihop network, it is much more difficult to achieve time synchronization, because nodes are spread out in multiple broadcast domains. IEEE 802.11 TSF does not work in a multihop network, mainly due to the reason that beacon messages sent on different broadcast domains do not agree with each other. As elaborated later, this leads to a problem we call *time partitioning* where the time in two groups of nodes can keep on drifting away from each other, even though they are connected.

Several clock synchronization protocols have been proposed for multihop wireless networks, including sensor networks. They are summarized in Section 4.2. However, as discussed in Section 4.2, these protocols do not support the synchronous operations well. Some protocols have low overhead but cannot avoid time partitioning problem, and other protocols that achieve global accuracy incur too much overhead.

Due to the difficulty of clock synchronization in a multihop network, protocol designers often avoid synchronous operations [39], or assume that the clocks are synchronized using out-of-band mechanisms such as GPS [31]. Also, some protocols introduce additional overhead to avoid relying on time synchronization. One example in this approach is STEM [40], which uses a second channel.

To support synchronous operations, we develop a protocol that maintains the clock error under a certain bound in a stable manner so that this bound can be used for other protocols running on top of this synchronization protocol. Thus, the protocol we propose in this chapter aims to achieve clock synchronization at a low cost.

In the rest of the chapter, we first define our problem formally, and discuss related work. Then, we describe our proposed clock synchronization protocol and present proofs on its features. Finally, we present and discuss experimental results based on simulations.

4.1 Problem Definition

In this section, we formally describe the problem addressed in this chapter. We also define terms and variables that will be used throughout the chapter.

We consider an ad hoc network that consists of multiple wireless nodes. The network may span multiple hops, meaning that a pair of nodes may be connected via other nodes acting as intermediate relays. No infrastructure exists. We assume that the network is always connected, meaning that when all nodes are active (not sleeping), one can find a path between any pair of nodes in the network.

Each node maintains a hardware clock. The value of the hardware clock is called *physical time* and the physical time of node i is denoted as T_i^P. We also assume that there exists a "real" clock, which represents the real time. The nodes have no knowledge of the real time. We denote the real time as t, and we can express the relationship between physical time of node i and real time as the following.

$$T_i^P = \alpha_{it} t + \beta_i \tag{4.1}$$

In Equation 4.1, both α_{it} and β_i are both determined by hardware clock and cannot be controlled by the protocol. Also, α_{it} may **change over time** due to temperature or other environmental changes.

Other than the hardware clock, each node also maintains a software clock. The value of the software clock is called *logical time*. Logical time can be modified by the protocol. The following equation describes the relationship between physical time and logical time. The logical time of node i is denoted as T_i.

$$T_i = T_i^P + \gamma_i \tag{4.2}$$

In the equation, γ_i is the parameter that can be controlled by the synchronization protocol. A node can correct the logical time to reduce the time difference with other nodes, and this process is called *time synchronization*. So when we say node A synchronizes to node B, it means that node A adjusts the γ value to reduce $|T_A - T_B|$. Using the above two equations, we can state the relationship between logical time and the real time as follows.

$$T_i = \alpha_{it}t + \delta_i \tag{4.3}$$

where $\delta_i = \beta_i + \gamma_i$. In the above equation, α is called the *clock rate*, and δ is called the *clock offset*. From here on, when we refer to the "time" at a node, we typically mean logical time at that node (the meaning should be clear from the context).

Suppose a set of nodes are deployed to form a network. Even if the nodes start their clocks exactly at the same time, after some interval, nodes will have different logical time because the nodes have different clock rates. In order to synchronize the time, each node exchanges messages telling their local time to other nodes. Then nodes use these messages to adjust their clocks so that the time difference among nodes is kept small. Due to the uncertainty in message delay, the nodes in the network cannot be synchronized to the exact same time. We define *clock error* to be the time difference between a pair of nodes, and we denote the clock error of node i and j as Δ_{ij}. Specifically,

$$|T_i - T_j| = \Delta_{ij} \tag{4.4}$$

Also, we define the maximum of all the clock errors as the *global clock error*, which is denoted as $\max \Delta_{ij}$. The goal of a synchronization protocol is to make the global clock error small so that it can be always kept under a certain threshold, which we call the *synchronization threshold.* The synchronization threshold represents the amount of accuracy an application requires. Specifically, we define our goal of time synchronization as follows. For a given synchronization threshold W,

$$\max \Delta_{ij} \leq W \tag{4.5}$$

If the protocol guarantees this upper bound on the global clock error, an application that runs on top of this protocol can use this information to set up a "margin" before starting a synchronized operation. For example, in a power management scheme, assume that the time is divided into beacon intervals, and node A and B are supposed to wake up at the beginning of each beacon interval and exchange messages. If A knows that $\Delta_{ij} \leq W$, then A can wait for W before transmitting a packet, after A starts a new beacon interval. This is depicted in Figure 4.1.

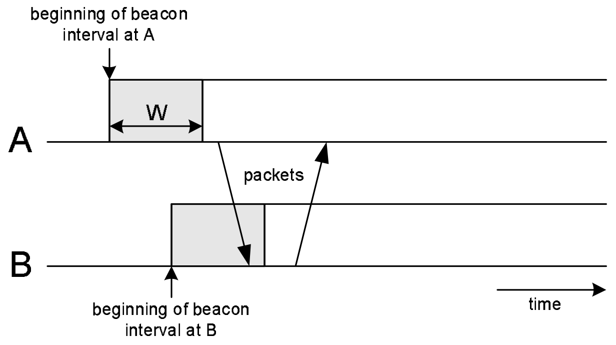

beginning of beacon interval at A
W
A
packets
B
beginning of beacon interval at B
time

Figure 4.1: Node A knows that the maximum synchronization error is W, so node A waits for duration W at the beginning of its beacon interval before transmitting.

4.2 Related Work

Numerous time synchronization schemes exist in the context of wired and wireless networks (including sensor networks) [41–48]. In this section, we review representative works in this area.

In wired networks, clock synchronization is often achieved using the Network Time Protocol (NTP) [41]. In NTP, multiple canonical sources are used as reference clocks, and a hierarchical structure is built that are rooted at these sources. Timestamped packets are exchanged along the branches of the trees, so that all nodes can synchronize to one of the canonical sources.

NTP is designed for wired networks, with the assumption that the network is mostly static. So NTP pre-configured synchronization hierarchy. However, in a wireless networks, topology may change frequently due to mobility, node failures and link failures. Also, predefined canonical sources may not exist in an ad hoc network. Thus NTP cannot be directly applied to wireless ad hoc networks.

For wireless LANs, IEEE 802.11 standard has a synchronization protocol called timing synchronization function (TSF) [5]. TSF is used for power saving mode (PSM) where every node has to wake up at the same time (beginning of a beacon interval) to exchange messages. At the beginning of a beacon interval, each node picks a random delay before transmitting a beacon. When a node transmits a beacon, all other nodes receives the beacon, suppress their beacon transmissions, and synchronize to the beacon sender using the timestamp included in the beacon. A node only synchronizes to a faster node (a node with a faster logical time) to avoid going back in time. IEEE 802.11 TSF is described in more detail in the next section, as it is relevant to our proposed protocol.

This scheme has the problem of *fastest node asynchronism*, identified by Huang and Lai [38]. Since a node only synchronizes to a faster node, the time of the fastest node (a node with the fastest logical time in the network) will keep drifting away from other nodes, unless it becomes the beacon transmitter. As the number of nodes increases, the chance that the fastest node transmits becomes smaller. Huang and Lai propose a simple modification to TSF called ATSP (Adaptive Timing Synchronization Procedure), to reduce the impact of fastest node asynchronism [38]. The idea is to have each node adjust their frequency of beacon transmission according to the received beacon messages. In each beacon interval, if a node receives a beacon with a faster clock, it reduces its beacon frequency. If not, then it increase its beacon frequency until it reaches the maximum. This scheme works well in wireless LAN. However, when these schemes are applied to multi-hop networks, the clocks might still drift away because of the *time partitioning* problem as explained in the next section.

In multi-hop networks, Biaz et al. [74] proved that the lower bound on the clock error depends on the diameter of the network, because of uncertainty in message delay. Sheu et al. [49] proposed a scheme called Automatic Self-time-correcting Procedure (ASP), that synchronizes clocks in a multi-hop network.. This protocol has two features. First, as in ATSP, the frequency of beacon transmission is adjusted according to the relative time values among neighbors, so that a faster node transmits beacon with higher probability than a slower node. Second, a node estimates the clock rate difference with its neighbor who is a faster node, so that it can automatically adjust its clock even when it does not receive a beacon from a faster node. This protocol improves the synchronization accuracy without incurring additional overhead compared to IEEE 802.11 TSF. Our proposed protocol does not use estimation on neighbor's clock rate, because it is not easy to obtain an accurate estimation. Clock rates may change due to environmental changes such as temperature change.

Fan and Lynch proposed an algorithm for synchronizing clocks in wireless networks [50]. Their algorithm ensures that the clock difference of two nodes that are d hops away in the network is bounded by a linear function of d at almost all times. Their algorithm requires at most one local broadcast per node in each synchronization "round". Our proposed protocol reduces the overhead further, so that the number of local broadcast *per broadcast domain* is almost one. Our proposed protocol ensures that the maximum clock difference is bounded by a linear function of the network diameter D, but it is also bounded by the hop distance between two nodes (d).

Ganeriwal et al. proposed a scheme similar to NTP, but modified to work in sensor networks [47]. The protocol, called Timing-Sync Protocol for Sensor Networks (TPSN), builds a tree structure in the network so that all nodes synchronizes to the root node. To synchronize a pair of nodes, TPSN uses pairwise message exchange to reduce estimation error. This protocol achieves good accuracy, but each synchronization round requires $2n$ transmissions, when the number of nodes is n. We aim to reduce the cost so that only a small portion of nodes do not broadcast a message in a round. The lightweight tree-based synchronization algorithm (LTS) [46] is similar to TPSN, but each node chooses the synchronization period based on the desired accuracy.

Until now, all protocols assumed that a node can stamp the time at the MAC layer, just before transmitting the packet. So the uncertainty in delay for contending the channel is removed. The Reference-Broadcast Synchronization (RBS) [43], considers this delay, because it assumes that the timestamping is done at a higher layer. To remove the uncertainty in the delay before a node gains channel access, the protocol proposes receiver-receiver synchronization paradigm. The idea is to have a reference node broadcast packet, and receivers compare their observations to synchronize time to each other. This protocol achieves high accuracy, but at a cost of high overhead. For multi-hop synchronization, they propose to use nodes in an overlapping area of two broadcast domains to transfer time information.

Li and Rus proposed three mechanisms for achieving clock synchronization in sensor networks [48]. In their diffusion-based algorithm, each node periodically reads its neighbors' clocks and computes average. Then it returns the average value to its neighbors. This scheme has a tradeoff between convergence time and overhead.

Finally, Elson and Estrin proposed the concept of *post-facto synchronization* [42]. In post-facto synchronization, the clocks are left unsynchronized. When an event happens, the relevant nodes coordinate with each other to figure out what event happened at what time. On the other hand, in a priori synchronization, nodes exchange messages to maintain clocks synchronized. Post-facto synchronization can preserve ordering of events, but cannot be used to support synchronous operations, because the clocks need to be synchronized prior to the operation.

Our protocol extends IEEE 802.11 TSF to work in multi-hop networks by having each node maintain a soft state. Since TSF is highly relevant to our protocol, we examine the protocol in a greater detail in the next section.

4.3 IEEE 802.11 Timing Synchronization Function (TSF)

In this section, we describe IEEE 802.11 TSF in detail, and discuss issues when this protocol is applied to a multi-hop network.

In IEEE 802.11 TSF, the synchronization takes place in every beacon period. At the beginning of a beacon period, each node waits for a random delay before transmitting a beacon. When a node transmits a beacon, the nodes that receives the beacon suppress their beacon transmissions. So for a wireless LAN, only a single packet is transmitted in each beacon period.

Before transmitting a beacon, the sender records the timestamp in the beacon packet using its clock. The timestamp is generated just before the node transmits the packet, so that the uncertainty in the MAC contention delay can be removed. When a node receives the beacon packet, it reads the timestamp and estimates the current time of the sender's clock considering the transmission and propagation delay. If the receiver has a slower time, it synchronizes to the sender by adjusting its time. Node do not synchronize to a slower node to avoid going back in time.

Suppose node A transmits a beacon and B receives it. If A's clock is faster than B's, B adjusts its time to match that of A. There is a delay between the point of time node A stamps its time in the beacon packet, and the time B receives the beacon. This delay consists of the transmission time and the propagation delay, as illustrated in Figure 4.2.

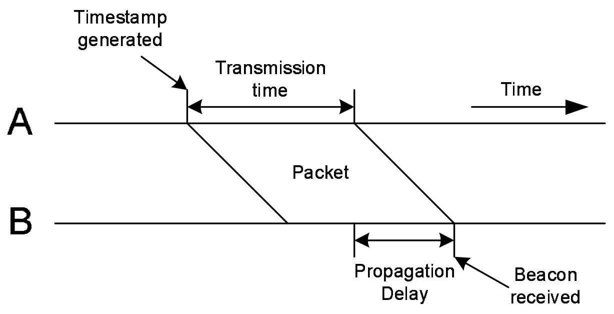


Figure 4.2: When node B receives a beacon from node A, it has to estimate A's current time considering the transmission time and the propagation delay.

The transmission time can be measured using transmission rate and packet size. Let B be the transmission rate and p be the packet size. Then the transmission time can be calculated as

$$T_t = p/B \tag{4.6}$$

Since we know the transmission rate and the packet size, we can obtain the accurate estimation of this delay. However, due to the difference between rate of the receiver's clock and rate of the real clock, an estimation error occurs. The error in estimated transmission time, ϵ_t, is

$$\epsilon_t = |\alpha_R - 1| \times T_t \tag{4.7}$$

where α_R is the receiver's clock rate. We assume that the clock rates are within the range

[0.9999, 1.0001] (±0.01%). Also, we assume that the size of a beacon packet is 56 bytes, which consists of 24 bytes of preamble, and other 32 bytes of data. Finally, we assume that the preamble is transmitted at 1Mbps, and data is transmitted at 2Mbps. Then, in Equation 4.7,

$$\max|\alpha - 1| = 0.0001 \tag{4.8}$$

and,

$$T_t = \frac{192}{10^6} + \frac{256}{2 \times 10^6} = 320\mu s \tag{4.9}$$

So the maximum estimation error for the transmission time is

$$\max \epsilon_t = 320 \times 0.0001 = 0.032\mu s \tag{4.10}$$

To estimate the propagation delay, we need to know the distance from the source to the destination. Since the distance is unknown, we use the upper bound as an estimate. Then the maximum estimation error for the propagation delay is

$$\max \epsilon_p = d_{max} \times \frac{1}{C} \tag{4.11}$$

where d_{max} is the maximum transmission range and C is the speed of light. If we assume d_{max} to be 250m, then the maximum error for propagation time would be approximately 0.8 μs.

On the whole, the maximum estimation delay, ϵ_{max} is

$$\epsilon_{max} = \max \epsilon_t + \max \epsilon_p \tag{4.12}$$

With 2Mbps of channel bandwidth, 56 bytes of packet size and the transmission range of 250m, the maximum error in synchronizing a pair of nodes is approximately 1 μs.

The IEEE 802.11 TSF is efficient in terms of communication cost, because only one packet is transmitted for each broadcast domain in each beacon period. However, when applied to multi-hop networks, TSF may fail to synchronize the clocks due to the *time partitioning* problem. Consider the scenario in Figure 4.3. Suppose node A is faster in time than B, and node D is faster than

C. Then node A and D have higher chance of transmitting beacons before node B and C. So if A and D transmit beacons, B synchronizes with A and C synchronizes with D. If this happens for several periods, the time between (A,B) and (C,D) will drift away unboundedly. We call this problem *time partitioning*, because even though these two groups of nodes are connected with each other, they do not exchange time information. If node A has a higher clock rate than D, for these two groups to synchronize to each other, B has to transmit a beacon, so that it can propagate to node C and then D. When the number of nodes increases, the problem of time partitioning has an significant impact on the clock accuracy, as shown by simulations in Section 4.5. Also, note that giving faster nodes higher chance in beacon transmission such as in ATSP increases the impact of the time partitioning problem.

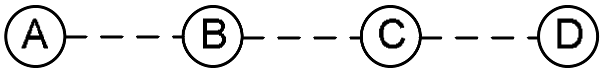


Figure 4.3: A simple chain topology with 4 nodes.

To prevent the time partitioning problem and maintain the clocks synchronized, we need to make sure that every node is synchronized with the fastest node within a certain period. Suppose in Figure 4.3, node A is the fastest node. Then for node D to synchronize with node A, the time information has to propagate through node B and C. So if we preserve the rule that only one node transmits in a broadcast region, node A has to transmit in the first beacon interval, node B in the next interval, and finally node C in the next interval for D to synchronize with A. This is similar to establishing a path between nodes, so that the packet is forwarded one hop at each beacon interval.

By maintaining a small amount of soft state at each node, we can establish an implicit path from the fastest node to all other nodes. This is what our proposed protocol does, and it is explained in detail in the next section.

4.4 Multi-hop Timing Synchronization Function (MTSF) [2]

The basic idea of the proposed protocol, MTSF, is to have each node maintain a path to the fastest node in the network, and propagate the time of the fastest node through the path, so that every node can synchronize with the fastest node within a certain period of time.

For example, consider the scenario in Figure 4.3. Suppose node A is the fastest node, and thus all other nodes need to synchronize to node A. In order for all nodes to synchronize with node A, node B must first synchronize with node A. After that node C should synchronize with node B, and finally node D should synchronize with node C. The propagation of clock information is illustrated in Figure 4.4. In MTSF, we try to achieve a schedule as in Figure 4.4 to have the clock information propagated from the fastest node to all other nodes in the network.

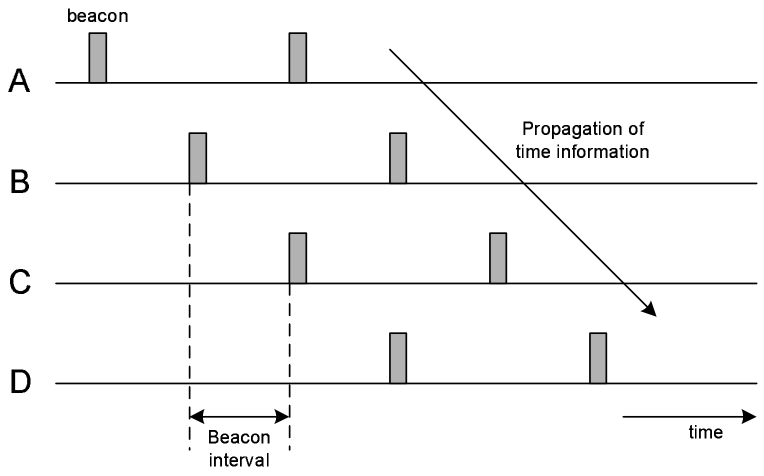


Figure 4.4: An example beacon transmission schedule.

In MTSF, each node schedules beacon transmission every two beacon intervals (a beacon interval is also called a *round*). At the beginning of a round in which the node schedules beacon transmission, it waits for a random delay before transmitting a beacon. As in IEEE 802.11 TSF, the sender stamps its time in the packet just before transmitting the packet. When a node receives the beacon, it compares the clock of the sender and itself considering the transmission time and propagation delay. If the sender's clock is faster, the receiver synchronizes to the sender. After receiving a beacon, a node decides if it should suppress its own beacon according to the rule explained later.

Every node maintains a "parent" variable, initially set to the identifier of itself. When a node finds a neighbor node with a faster clock, it sets the "parent" variable to be the identifier of the faster node. If there are multiple faster nodes in the neighborhood, the fastest node among those neighbors are chosen as the parent node.

Once the parent node is chosen, the node schedules its beacons in the rounds that its parent node does not schedule beacons. So if the parent node schedules beacons on "odd-numbered" rounds, the child node schedules beacons on "even-numbered" rounds.

Using this simple scheme, each node eventually establishes a path towards the fastest node. When every node in the network has a path between itself and the fastest node, we say that the protocol has converged to a *steady state*. We will prove that starting from an arbitrary state where each node has an arbitrary time, this protocol converges to the steady state. For simplified analysis, the analysis assumes that during the process of convergence, packets sent are correctly received (no packet loss), network topology does not change, and the clock rates of nodes do not change. (So the clock rates are denoted as α_i in the analysis, instead of α_{it}.) In Section 4.5, we also study the impact of packet loss, temporal changes in clock rates, and mobility on the performance of MTSF.

Before proving this self-stabilization property, we first derive the upper bound on the clock error between a node and the fastest node in the network given that the protocol is in a steady state. The result is used to prove the self-stabilization property of MTSF.

Lemma 1 The protocol reaches a steady state, and the upper bound on the global clock error is $2f(D+1)L + D\epsilon_{max}$, where f is maximum rate difference compared to the real clock, D is the network diameter, L is the length of a beacon interval, and ϵ_{max} is the maximum estimation error.

Proof Suppose the fastest node in the network is R, and the node we want to calculate the clock difference is k hops away from R, in the path established by the protocol. Note that k may not necessarily be the shortest hop distance between these two nodes. We name the node C_k, and the nodes in the path from R to C_k are C_1, C_2, ..., C_{k-1}.

To make the equations simple, we ignore the impact of estimation error ϵ_{max} (defined in Section 4.3 in the analysis). The effect of ϵ_{max} is added to the upper bound at the end of the analysis.

At ith beacon interval, node R sends a beacon to node C_1, and C_1 synchronizes to node R. After the synchronization, $T^i_{C_1} = T^i_R$. Then at the beginning of $(i+1)$th interval, T_{C_1} can be expressed as follows.

$$T^{i+1}_{C_1} = T^i_R + \alpha_{C_1} L \tag{4.13}$$

where α_{C_1} is the clock rate of C_1 and L is the length of the beacon interval. Now at $(i+1)$th interval, C_1 sends a beacon and C_2 is synchronized to C_1. Since C_1 is the fastest neighbor of C_2, C_1 is the last node that C_2 synchronizes to in the beacon interval. Then at the beginning of $(i+2)$th interval, the time of C_2 is

$$T_{C_2}^{i+2} = T_{C_1}^{i+1} + \alpha_{C_2} L = T_R^i + (\alpha_{C_1} + \alpha_{C_2})L \tag{4.14}$$

Continuing this process, node C_{k-1} will send a beacon at $(i+k-1)$th beacon interval, and C_k will synchronize to C_{k-1}. Then at the beginning of $(i+k)$th beacon interval, the time of C_k is

$$T_{C_k}^{i+k} = T_R^i + (\sum_j \alpha_{C_j})L \tag{4.15}$$

where $j = 1, 2, \ldots, k$. Also, since node R never synchronizes to other nodes, at the beginning of $(i+k)$th beacon interval,

$$T_R^{i+k} = T_R^i + \alpha_R k L \tag{4.16}$$

Thus, the clock difference between C_k and R at the beginning of $(i+k)$th beacon interval is

$$\begin{aligned} \Delta_{C_k R} = T_R^{i+k} - T_{C_k}^{i+k} &= \alpha_R k L - (\sum_j \alpha_{C_j})L & (4.17) \\ &= (\sum_j (\alpha_R - \alpha_{C_j}))L & (4.18) \end{aligned}$$

where $j = 1, 2, \ldots, k$.

Since each node sends a beacon in every other beacon interval, C_{k-1} would not send a beacon at $(i+k)$th beacon interval. Thus, C_k does not synchronize to C_{k-1} in $(i+k)$th beacon interval. So at the end of $(i+k)$th beacon interval, the clock difference between C_k and R becomes the maximum, and it is

$$\Delta_{C_k R} = (\sum_j (\alpha_R - \alpha_{C_j}) + (\alpha_R - \alpha_{C_k}))L \tag{4.19}$$

This is the maximum clock error between a node and the fastest node in the network.

Suppose D is the network diameter, which is the maximum hop distance between any pair of nodes in the network. Then the maximum clock error among all pairs of nodes in the network will be

$$\max \Delta = (\sum_j (\alpha_R - \alpha_{C_j}) + (\alpha_R - \alpha_{C_D}))L \tag{4.20}$$

where j = 1, 2, ..., D, and C_1, C_2, ..., C_{D-1} are the nodes in the path from R to C_D.

Since the clock rates are unknown, a node cannot precisely determine $\max \Delta$. So we can consider the worst case, where R has the maximum clock rate and all other nodes have the minimum allowable clock rate. If the clock rate is required to be within the range $[1-f, 1+f]$, then $\max \Delta$ becomes

$$\max \Delta = (\sum_j (\alpha_R - \alpha_{C_j}))L = 2f(D+1)L \tag{4.21}$$

Now we take into account ϵ_{max}. Since the maximum estimation error for each hop is ϵ_{max}, the new upper bound on the network synchronization error is

$$\max \Delta = 2f(D+1)L + D\epsilon_{max} \tag{4.22}$$

If we assume f is 0.0001, D is 10, L is 100ms, and ϵ is 1μs, then the maximum network synchronization error will be 230 μs.

This is a very conservative calculation of the global clock error because we assumed that the fastest node has the maximum clock rate, and all other nodes have the minimum clock rate. Thus, in reality, MTSF may achieve a lower bound on the accuracy in the steady state. The simulation results show that MTSF achieves a clock error of much less than 230 μs in average case.

Now we prove that the protocol converges to a steady state, where every node in the network establishes a path to the fastest node.

Lemma 2 Starting from an arbitrary state, the protocol eventually enters a steady state, where every node in the network establishes a path to the fastest node. In the steady state, every node updates its time in every two beacon intervals to match the rate of the fastest node.

Proof If node S has node D as its parent, we say that node S "points to" node D. Also, if node S can reach node R by going up the path, we say that node S "points towards" node R.

We prove that if a node always chooses the fastest neighbor as its parent, it eventually points towards the fastest node in the network. When every node points towards the fastest node, the protocol enters a steady state.

We start with a simple example and generalize the argument to any possible cases.

Consider the scenario in Figure 4.5. The clock rate of node A, B, C, D and E is α_A, α_B, α_C, α_D, and α_E, respectively. Suppose $\alpha_A > \alpha_E > \alpha_B > \alpha_D > \alpha_C$. At some point of time, B will regard A as the fastest node in the network, because node A has a faster rate than B. On the other hand, D will regard E as the fastest node. After that, node C has to decide whether it should pick node B or node D as its parent. We argue that if C always chooses a faster node as its parent, C will eventually choose B as its parent. Applying the same argument, D and E will also eventually point toward node A.

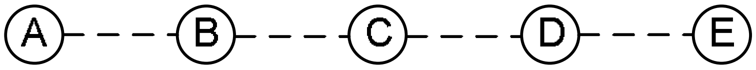


Figure 4.5: A network scenario with 5 nodes placed in a chain topology.

To show this, we consider the clock difference Δ_{BC} and Δ_{CD} at the start of kth beacon interval. Considering all possible situations, we show that eventually Δ_{BC} becomes larger than Δ_{CD} so that node C chooses B as its parent. For simplicity of the analysis, we ignore the impact of the estimation error, ϵ_{max}. However, the argument below still holds even if the estimation error is taken into account.

Let L be the length of a beacon interval. Assume that in $(k-1)$th beacon interval, node A broadcasts a beacon, and B synchronizes to A. So at the end of $(k-1)$th beacon interval,

$$\Delta_{AB} = (\alpha_A - \alpha_B)L \tag{4.23}$$

Thus,

$$T_B = T_A - (\alpha_A - \alpha_B)L \tag{4.24}$$

At the kth beacon interval, node B transmits a beacon and node C synchronizes to node B. When node C receives node B's beacon, the clock difference between node B and C is

$$\Delta_{BC} = T_B - T_C = T_A - (\alpha_A - \alpha_B)L - T_C \tag{4.25}$$

Since $T_A = \alpha_A t + \delta_A$ and $T_C = \alpha_C t + \delta_C$,

$$\Delta_{BC} = \alpha_A t + \delta_A - (\alpha_A - \alpha_B)L - (\alpha_C t + \delta_C) \tag{4.26}$$

We can rewrite the equation as

$$\Delta_{BC} = (\alpha_A - \alpha_C)t + u \tag{4.27}$$

where u is a constant ($u = \delta_A - (\alpha_A - \alpha_B)L - \delta_C$).

Assume that the clock rate of node E is faster than C, but is not faster than A ($\alpha_A > \alpha_E > \alpha_C$). Similar to how C synchronizes with A, C will also synchronize to E using messages propagated from E via D. When node C receives a beacon from node D, the clock difference between C and D is

$$\Delta_{DC} = (\alpha_E - \alpha_C)t + v \tag{4.28}$$

where v is a constant ($v = \delta_E - (\alpha_E - \alpha_D)L - \delta_C$). Since $\alpha_A > \alpha_E$, as t increases, eventually Δ_{BC} becomes larger than Δ_{DC}. Thus, node C eventually chooses node B as its parent.

We can generalize this argument and prove that every node in the network will eventually choose its parent towards the fastest node, if each node chooses the fastest neighbor as its parent.

We prove this using the previous argument and by induction on hop distance of a node from the fastest node. Let R be the fastest node in the network. For the base case, suppose node A_1 is a one-hop neighbor of node R. Since node R is the fastest node, for any node i in A_1's neighbor set.

$$\Delta_{A_1 R} > \Delta_{A_1 i} \tag{4.29}$$

where $\Delta_{A_1 R}$ is the clock difference between A_1 and R, and $\Delta_{A_1 i}$ is the clock difference between A_1 and node i. (We omit the case where there are multiple fastest nodes. In this case, multiple trees can be established in the network. Nevertheless, the upper bound on the clock error will be the same as the case where there is a single fastest node.)

Now suppose node A_{k+1} is $k+1$ hops away from node R. It has a neighbor node A_k, which is already pointing towards node R. All of A_k's ancestors, A_1, A_2, ..., A_{k-1}, are pointing toward node R.

Since the nodes A_i $(i = 1, 2, ..., k)$ are already pointing toward node R, from Equation 4.19, the maximum of $\Delta_{A_k R}$ at the start of a beacon interval is

$$\Delta_{A_k R} = (\sum_i (\alpha_R - \alpha_{A_i}) + (\alpha_R - \alpha_{A_k}))L \tag{4.30}$$

where $i = 1, 2, ..., k$. Thus,

$$\Delta_{A_{k+1} A_k} = T_R - ((\sum_i (\alpha_R - \alpha_i) + (\alpha_R - \alpha_{A_k}))L) - T_{A_{k+1}} \tag{4.31}$$

Since $T_R = \alpha_R t + \delta_R$ and $T_{A_{k+1}} = \alpha_{A_{k+1}} t + \delta_{A_{k+1}}$, $\Delta_{A_{k+1} A_k}$ can be written as

$$\Delta_{A_{k+1} A_k} = (\alpha_R - \alpha_{A_{k+1}})t + c \tag{4.32}$$

where c is a constant $(c = \delta_R - ((\sum_i (\alpha_R - \alpha_i) + (\alpha_R - \alpha_{A_k}))L) - \delta_{A_{k+1}})$. Similarly, if another neighbor of node A_{k+1}, B, is pointing towards node S, then

$$\Delta_{A_{k+1} B} = (\alpha_S - \alpha_{A_{k+1}})t + c' \tag{4.33}$$

$(c = \delta_S - ((\sum_i (\alpha_S - \alpha_i) + (\alpha_S - \alpha_{A_k}))L) - \delta_{A_{k+1}})$. Since α_R is greater than α_S, eventually node A_{k+1} will choose A_k as its parent.

So if every node always chooses the fastest neighbor as its parent, then eventually every node will point towards the fastest node in the network, and the protocol enters a steady state.

Until now, we have shown that if each node transmits beacon in every other beacon interval and choose the fastest neighbor as its parent, all nodes will eventually point towards the fastest node in the network. However, if every node transmits beacon in every other beacon interval, the communication overhead of this protocol is proportional to the number of nodes, which is not scalable.

The reason for having each node maintain the “parent” variable is to reduce the communication overhead while still preserving the upper bound on the global clock error. We define *leaf node* to be a node which does not have any child that is pointing towards that node. The leaf nodes do not contribute to the accuracy of the protocol, so they do not need to transmit beacons every other beacon interval.

However, if a leaf node does not transmit beacon at all, the protocol will not be able to adapt to change in the topology. Suppose a new node joins the network and it needs to point to the leaf node to reach the fastest node. If a leaf node does not transmit at all, the new node would not be able to synchronize to the fastest node. Thus, leaf nodes can transmit at a low frequency to advertise their existence, but not in every other beacon interval.

To identify whether a node S is a leaf node or a non-leaf node, we need feedback from the nodes who consider S as their parent. So every node includes its parent identifier in the beacon packet, so that when a node receives a beacon with the parent identifier as itself, then the node knows that it is a non-leaf node.

The beacon transmission rules for non-leaf nodes and leaf nodes are as follows. If a node is a non-leaf node, it transmits beacons in every other beacon interval, regardless of whether it receives a beacon in that interval or not. If a node is a leaf node, it suppresses its beacon if it receives a beacon from *another leaf node with the same parent* in the beacon interval. This is to make sure that their parent node is notified of their existence. In addition, to guarantee that every leaf node transmits a beacon eventually, we force every leaf node to transmit beacon with a probability p_t even though they receive a beacon from another node sharing the same parent. If a non-leaf node does not receive beacons from any of its children for several consecutive beacon intervals, it regards itself as a leaf node. (In the simulations, a node regards itself as a leaf node if it does not receive a beacon from a child in 4 consecutive beacon intervals.)

This scheme reduces the communication cost of MTSF significantly, as will be shown in the simulation results. As we will see in the next section, the number of non-leaf nodes grows slowly as the number of nodes increase in a given area. So the communication overhead of MTSF grows slowly with increasing node density, which makes MTSF a scalable protocol.

4.5 Performance Evaluation

In this section, we report results from the simulations we performed to study the performance of MTSF. For the simulations, we have used our own simulator written in C++. Through simulations, we expect to see if MTSF successfully bounds the clock error between any pair of nodes, and if MTSF achieves this goal at a low cost. Also, the analysis in Section 4.4 assumed that during the process of convergence, there is no packet loss, the clock rates do not change, and there is no changes in the topology due to mobility. In the simulations, we consider these factors and study how each factor impacts the performance of MTSF.

For comparison, we also simulate a modified version of IEEE 802.11 TSF. In the modified version of IEEE 802.11, when a node receives a beacon, it does not always suppress its beacon, but transmits the beacon with a probability p. If p is 0, then the protocol falls back to the basic IEEE 802.11 TSF. If p is 1, then every node transmits a beacon in a beacon period.

To measure the performance of a protocol in terms of clock accuracy, we use the following metrics.

- Global clock error: This is the maximum of clock difference between any pair of nodes in the network. We trace the maximum clock error over time to see the behavior of the protocol. Tracing maximum clock error also shows how fast the protocols converge, because at the beginning of each simulation, the clock values are arbitrarily chosen for each node.

- Percentage of time that the network is out of synchronization: For different threshold values, we measure the percentage of time the maximum clock error exceeds the threshold.

To measure the protocol overhead, we use the following metrics.

- Average number of beacon transmissions per round in a broadcast domain: For each round, we measure how many beacons are transmitted in a broadcast region. For example, if a node receives three beacons in a round, the number of beacons transmitted in that broadcast region is 3. It is averaged over all nodes and over the whole simulation time.

- Percentage of leaf nodes in the converged synchronization tree: This metric indicates how MTSF builds an efficient synchronization tree in the network.

We measure these metrics with different network sizes, packet loss rate, maximum clock rate change and maximum speed to see the impact of these factors. We first describe our simulation setup, and then we present and discuss the results.

4.5.1 Simulation Setup

In all of our simulations, nodes are randomly placed in a square-shaped region. The size of the area is 1000m $\times$ 1000m, unless otherwise specified. Every node has a fixed transmission range of 250m. Unless otherwise specified, we place 100 nodes in the simulation area, and their clock rates and initial clock values are randomly assigned.

The initial clock rates are randomly selected from the range [0.9999, 1.0001]. So after clocks are synchronized, the maximum clock error between two nodes after 1 second is $200\mu s$. The initial clock value for each node is randomly chosen from the range [0, 1000] milliseconds. The beacon period is 100 milliseconds, unless otherwise specified. The probability that a leaf node will transmit a beacon, p_t, is set to 0.2. If a non-leaf node does not receive a beacon from a child for 4 consecutive beacon intervals, the node regards itself as a leaf node.

Under these assumptions, we vary the packet loss rate to see the impact on the performance. If the packet loss rate is p_l, then a node transmits a packet, its neighbor successfully receives the packet with probability $1 - p_l$. Each receiver follows the fixed packet loss rate independent of other receivers.

Finally, the total simulation time is 1000 seconds for every simulations.

4.5.2 Simulation Results

Now we present and discuss our simulation results. As mentioned before, we evaluate our proposed protocol, MTSF, as well as IEEE 802.11 TSF with different probability that a node is forced to transmit beacon in a beacon period.

Accuracy

In the first simulations, we plot the global clock error over the whole simulation time. Figure 4.6 shows the result for different protocols. Figure 4.7 plots the same graph, but with a smaller scale on the Y-axis. We can see that with MTSF, the maximum clock error is always bounded under a certain threshold. With the original IEEE 802.11 TSF, the clocks may drift away until the fastest node gets a chance to transmit the beacon. With increased probability of forced transmission, the accuracy of TSF increases, and the accuracy of TSF with forced transmission probability 0.4 is comparable to the accuracy of MTSF. We will see later that the number of packets transmitted is much smaller with MTSF.

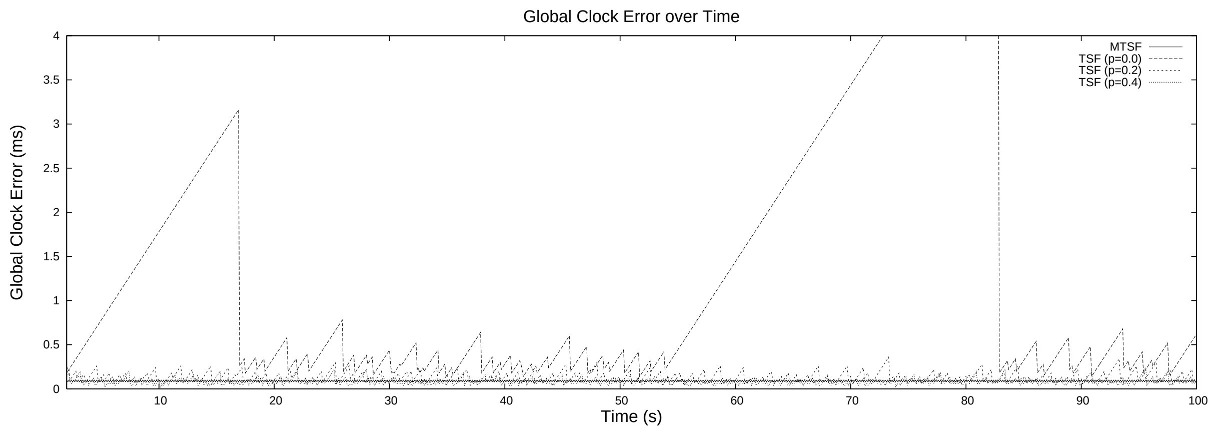


Figure 4.6: Global clock error over time. The global clock error is the maximum clock difference between any pair of nodes in the network.

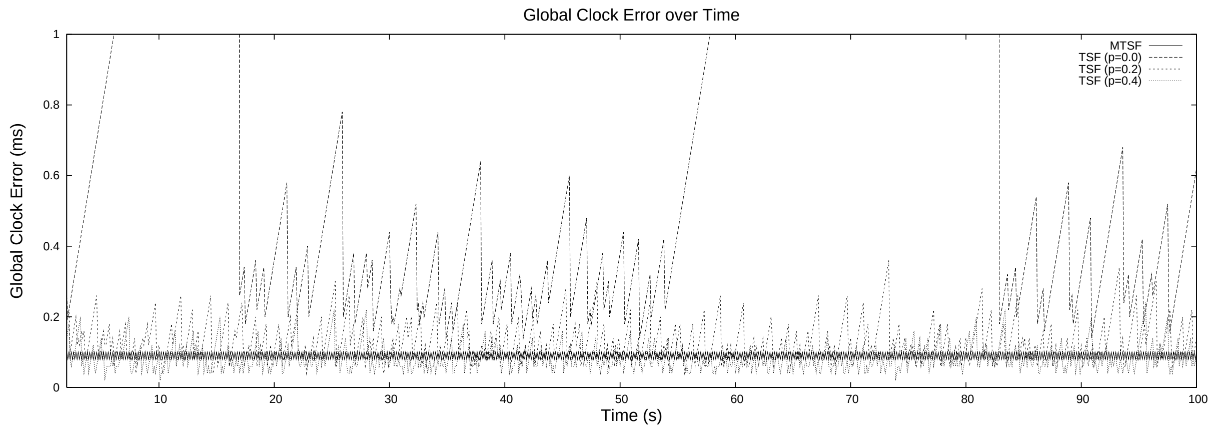


Figure 4.7: Global clock error over time. Shown with a smaller scale on the Y-axis.

If the global clock error exceeds a given threshold, we say that the network has become unsynchronized. To observe the accuracy of the protocols, we plot the percentage of time that the network is unsynchronized, for different thresholds. Figure 4.8 also shows that the accuracy that MTSF achieves is comparable to TSF with forced transmission probability between 0.2 and 0.4.

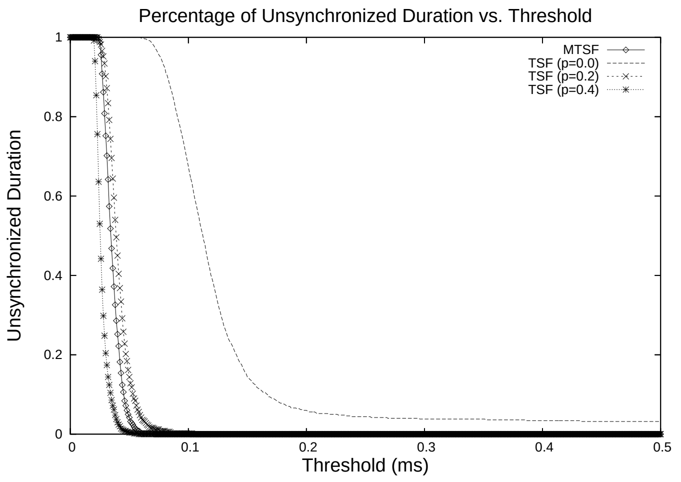


Figure 4.8: Percentage of time the network is unsynchronized. The network is considered as unsynchronized when the global clock error exceeds the given threshold.

Convergence

In addition to performance in terms of accuracy, we want to see how fast the network converges to a synchronized state using MTSF. Note that the initial values are assigned randomly from the range [0, 1000]. So initially, the clocks are unsynchronized by the maximum of 1 second.

Figure 4.9 plots the global clock error over time with different protocols, but it only shows the initial part where the clocks are being synchronized. For IEEE 802.11 TSF, the convergence time is longer when the forced transmission probability is low. As the probability increases, the convergence time decreases dramatically. For MTSF, the convergence time is comparable to TSF with forced transmission probability 0.2. The overhead of MTSF is lower than that of TSF with probability 0.2, as seen later.

Protocol Overhead

Next, we study the overhead of the protocols. To see the communication overhead, we measure the average number of beacons sent or received per round in a broadcast domain.

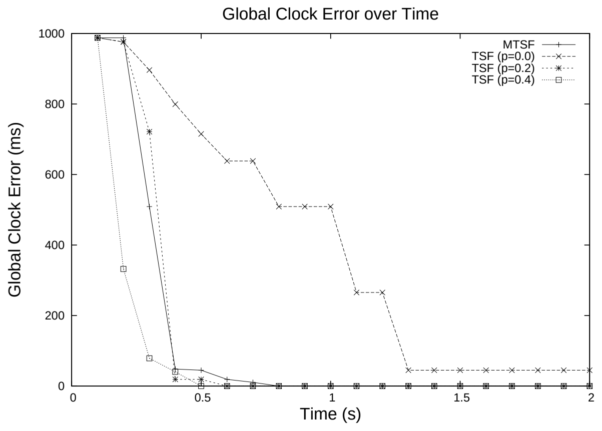


Figure 4.9: Global clock error over time. This graph shows the initial duration of time to synchronize clocks from initial state where clock values are randomly chosen from the range (0, 1000).

Figure 4.10 shows the result. The message overhead of MTSF is less than TSF with forced transmission probability 0.2. Moreover, as the number of nodes increase, the increase rate of message overhead is much slower than TSF. This result indicates that MTSF is scalable.

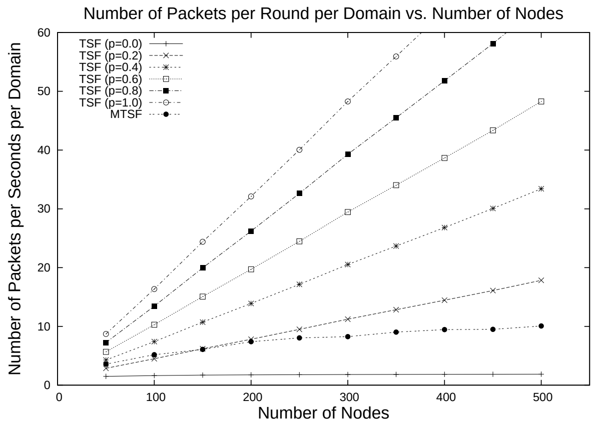
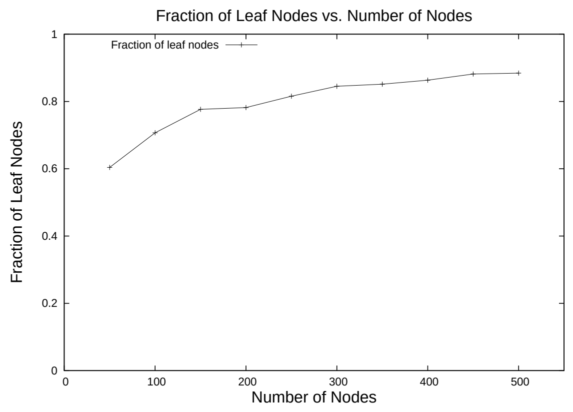
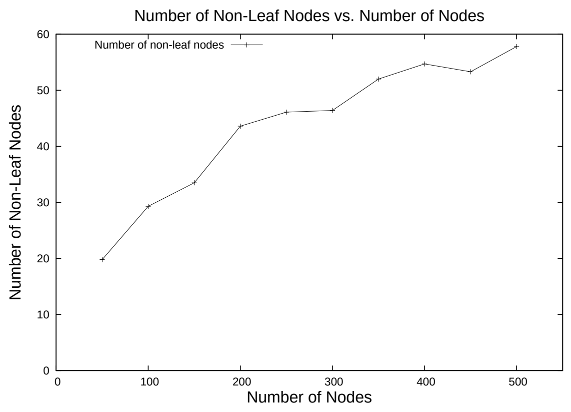


Figure 4.10: Protocol overhead of the protocols. This graph shows the average number of messages per second per domain.

The reduction in communication cost comes from building an efficient spanning tree in the network, with the fastest node as the root. If the percentage of leaf nodes is higher, then the communication cost is lower. To see how MTSF does well in terms of building a spanning tree, we measure the fraction of leaf nodes in a converged network. As shown in Figure 4.11(a), the fraction of leaf nodes increases as the number of nodes increase. Also, Figure 4.11(b) shows that the number of non-leaf nodes increases slowly as the number of nodes increases.



(a) Fraction of Leaf Nodes



(b) Number of Non-leaf Nodes

Figure 4.11: The leaf nodes in the synchronization tree built by MTSF protocol.

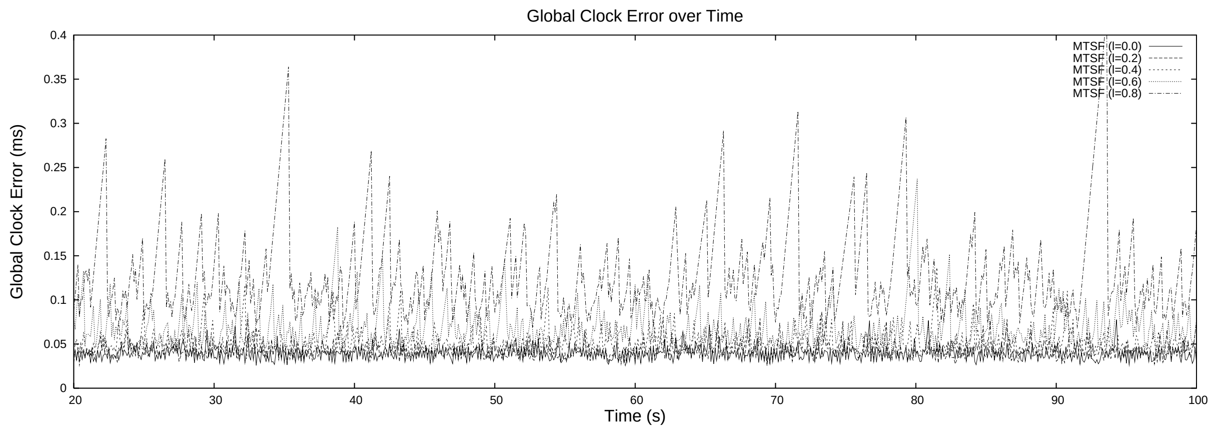


Figure 4.12: Impact of packet loss on the performance of MTSF. This graph shows the global clock error over time.

In Section 4.4, we have proved that an upper bound of global clock error is guaranteed. The proof assumed that there is no packet loss, and clock rates do not change over time. However, in reality, there can be packet loss due to bad channel quality or collisions. Also, clock rates may change over time, due to temperature or other environmental changes. If there is mobility, the tree structure in the network need to change over time. In the next set of simulations, we study the impact of packet loss, clock rate change, and mobility on the accuracy of MTSF. In terms of overhead, these factors had negligible impact on the amount of protocol overhead.

Impact of Packet Loss

In this simulation, We study how MTSF performs under different packet loss rates. As mentioned earlier, we assign a packet loss rate p_l, that is used for every packet. So when a packet is transmitted, it is received with probability $1 - p_l$. Figure 4.12 and Figure 4.13 shows the degradation in accuracy of MTSF when there is packet loss. The results show that MTSF can tolerate packet loss well when the packet loss rate is not too high. This tolerance of MTSF comes from the redundancy in which a node may receive multiple clocks from neighbors in a round and synchronize to them. So even when the beacon from the parent is lost, there may be other neighbors that transmit beacons in the same round. If the beacon holds a faster clock, the node can synchronize to the beacon and reduce the clock difference.

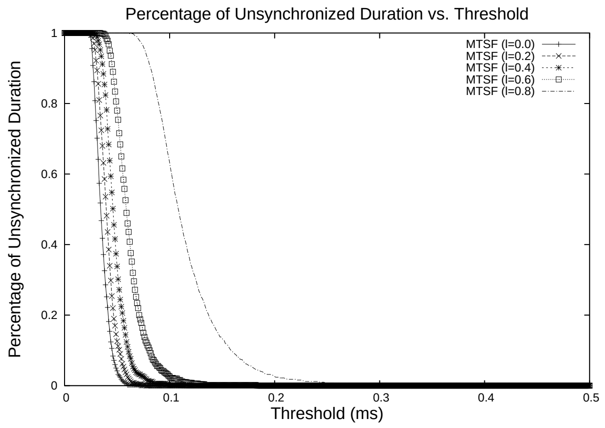
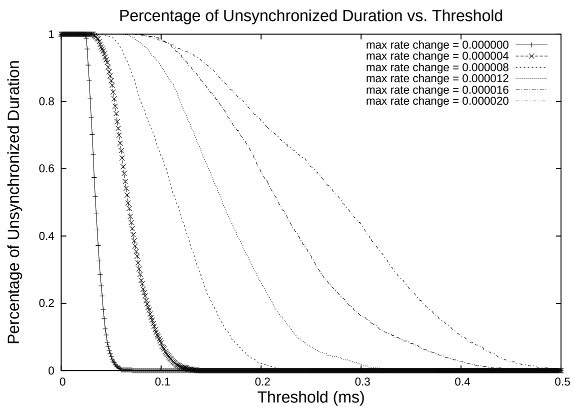


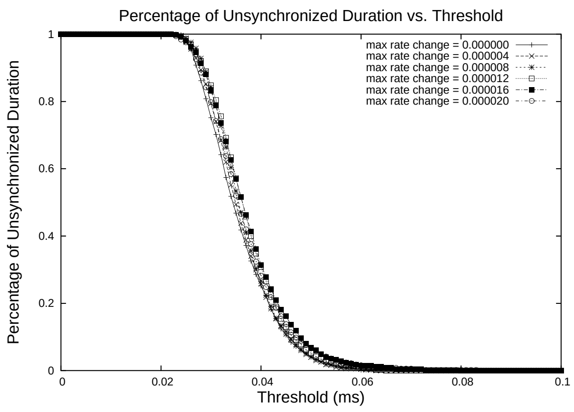
Figure 4.13: The accuracy of MTSF under scenarios with packet loss. This graph shows the percentage of time the network is unsynchronized.

Impact of Clock Rate Change

Now we study the accuracy of MTSF when clock rates change over time. Initially the clock rates are chosen uniformly random from the range [0.9999, 1.0001]. After each beacon interval (100ms in our simulations), the clock rate of each node may increase, decrease, or stay the same. The parameter we use here is the *maximum rate change*, which is the maximum amount that a node's clock rate may change in each beacon interval. For example, if a node's clock rate is 1.0000 in the current interval and the maximum rate change is 0.000004, in the next interval, the new clock rate is chosen from the range [0.999996, 1.000004].



(a) Clock rate unbounded



(b) Clock rate bounded by (0.9999, 1.0001)

Figure 4.14: Accuracy of MTSF under scenarios with temporal change in clock rates. The graphs show the percentage of time the network is unsynchronized.

In the first simulation, the clock rates are unbounded, so they may go out of the range [0.9999, 1.0001]. (For example, if the current clock rate is 1.0001 and the maximum rate change is 0.000004, then the clock rate in the next interval can be 1.000104.) Figure 4.14(a) shows the accuracy of MTSF under this scenario. The result shows that the clock accuracy drops almost linearly as the maximum rate change increases. This is because the maximum difference of clock rates between any pair of nodes in the network increases linearly. According to Equation 4.22, the accuracy decreases linearly when the maximum rate difference (f in the equation) increases. In the next simulation, we have bounded the clock rate to be within the range [0.9999, 1.0001]. (For example, if the current clock rate is 1.0001, and the amount of rate change is chosen to be 0.000002, the clock rate for the next beacon interval stays as 1.0001 instead of becoming 1.000102.) The result is shown in 4.14(b), and we can see that the accuracy is similar even if maximum rate change is varied. So it is not the clock rate change that significantly affects the accuracy, but it is the maximum difference of clock rates between nodes that decides the accuracy of MTSF.

Impact of Mobility

Finally, we study the impact of mobility on protocol accuracy. In the simulations, nodes move around according to the Random Waypoint Model [69]. We vary the maximum speed of the nodes.

Figure 4.15 shows the accuracy of MTSF when there is mobility. The result shows that even with high mobility at a maximum speed of 30m/s, the accuracy stays almost the same. This is because even with 30m/s of speed (67.5 miles/hour), the topology changes slow enough for the protocol to maintain an accuracy comparable to that under scenarios with no mobility.

From the simulations, we can see that MTSF achieves good accuracy even under packet losses, and the accuracy is achieved at a much lower cost than IEEE 802.11 TSF with forced transmission probability tuned to match the accuracy with MTSF. Also, we have seen that MTSF is scalable, because the overhead of MTSF increases slow as the node density increases. MTSF can tolerate packet loss well when the packet loss is not too high. Also, MTSF works well under scenarios with mobility or temporal changes of clock rates.

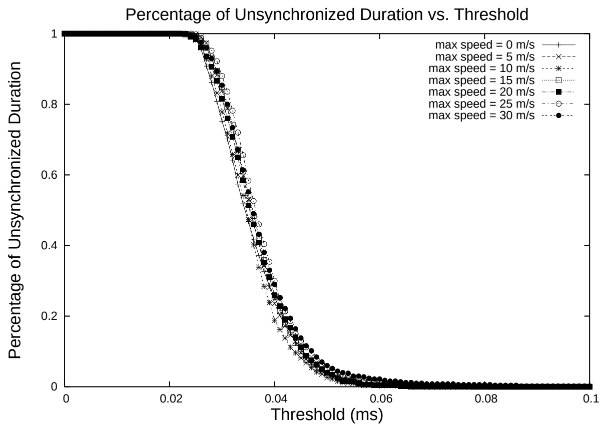


Figure 4.15: Accuracy of MTSF under scenarios with mobility. The graph shows the percentage of time the network is unsynchronized.

4.6 Summary

In this chapter, we have proposed MTSF, a time synchronization protocol for multi-hop wireless networks. Since MTSF is designed to support synchronous operations for applications, or other protocols running on top of MTSF, it aims to achieve stability in maintaining clock accuracy. At the same time, MTSF aims to reduce the communication cost used for synchronization. Reducing cost is important in achieving scalability.

Since a node only synchronizes to a faster node, all the nodes must synchronize with the fastest node in the network to avoid fastest node asynchronism. MTSF achieves this by implicitly building up a synchronization tree rooted at the fastest node in the network. During a short period of time when there is no change in network topology, the protocol quickly converges to a steady state. In the absence of packet loss, the protocol in the steady state guarantees an upper bound on the global clock error. The protocol tolerates packet losses when the loss rate is not too high. When the topology changes, the protocol quickly adapts to the new topology, without any explicit procedures. We have shown that the network converges to a steady state once the network topology is fixed, and also calculated the upper bound on the global clock error in the steady state.

Also, using the tree structure, MTSF reduces the number of beacon transmissions in each period. This is important because the synchronization process should not harm the performance of applications that use the synchronization service, by occupying significant amount of bandwidth.

The simulation results show that MTSF achieves stable clock accuracy at a low cost. So MTSF can efficiently support synchronous operations which is an important requirement for many applications and protocols. In the next chapter, we integrate MTSF with MMAC to study the performance of MMAC in a more realistic scenario where clocks are not synchronized.

Chapter 5

Integrating MMAC with Clock Synchronization and Routing

In Chapter 3 we proposed MMAC, a MAC protocol which uses periodic channel negotiation to utilize multiple channels. In MMAC, all nodes must switch to a common channel at the beginning of each beacon interval in order to start channel negotiation. Thus, clocks must be synchronized. In the evaluation of MMAC presented in Chapter 3, it was assumed that clocks are always synchronized. However, in practice, the rate of clocks can never be exactly the same, and also the clock speed can change dynamically according to environmental reasons such as temperature. In order to maintain clock synchronization, nodes must synchronize their clocks periodically by exchanging messages. In Chapter 4, we proposed MTSF, a light-weight clock synchronization service that can synchronize clocks in a multi-hop network. In this chapter, we integrate the clock synchronization protocol with MMAC, and see if MMAC can work in a real environment where clocks may drift.

MMAC is a link layer protocol, designed to work with any routing protocol. In Chapter 3, we used AODV as a routing protocol on top of MMAC. AODV tries to find a route which has the shortest hop distance from the source to the destination. In multi-channel networks, there can be metrics other than hop distance that can significantly affect the network performance. As we will see in this chapter, especially when running MMAC, it is important to find routes that do not overlap with each other. In this chapter, we identify issues in integrating routing and MMAC, and propose an extension to AODV that can improve the network performance.

Basically, the purpose of this chapter is to integrate all the protocols into a system, and evaluate their performance as a whole in a multi-channel multi-hop wireless environment.

5.1 Integrating MTSF with MMAC

The MTSF protocol, presented in Chapter 4, can be easily integrated with MMAC. In MMAC, all nodes switch to a common channel at the beginning of a beacon interval, and negotiate channels by exchanging ATIM messages. During the negotiation window, a node sends out an ATIM message only if it has packets to send during that interval. The synchronization message can also be broadcasted during the negotiation window, because all nodes will be listening on a common channel. To further reduce overhead, if a node is to send an ATIM packet, it can include timestamp in the ATIM packet rather than transmitting another packet for the purpose of synchronization. The behavior of MTSF on top of MMAC is as follows.

- At the beginning of each beacon interval, switch to the predefined common channel.
- If the node has an ATIM packet to send, do not schedule a beacon packet. The ATIM packet acts as a beacon carrying time information.
- If the node is not sending an ATIM packet and has to send a beacon packet according to the rules of MTSF (refer to Chapter 4), then schedule a beacon packet.

The states at each node is maintained according to MTSF rules. The MTSF protocol synchronizes clocks within a bound which depends on the synchronization interval and maximum hop distance of the network. If a node immediately sends out a beacon packet at the beginning of a beacon inveral, nodes that have slower clocks may not receive the packet. Thus, each node must wait for a delay longer than expected maximum clock error before transmitting a beacon packet or an ATIM packet. If the clock accuracy is within $\pm 0.01\%$ (as specified in IEEE 802.11 standard), beacon interval is 100 ms, and the maximum hop distance is 10, then the expected maximum clock error can be calculated according to Equation 4.22 in Chapter 4.

$\max \Delta = 2f(D+1)L = 2 \times 0.0001 \times (10+1) \times 0.1 = 220\mu s$

If the channel negotiation window is 20 ms, then this delay overhead is approximatedly 1%.

It is possible that the clocks have already drifted to the extent that the channel negotiation windows do not overlap. In this case, messages that include timestamps cannot be received properly, because the receiver might have already finished negotiation window and switched to another channel. To make nodes synchronize with each other eventually, we apply the following rule. At each beacon interval, nodes decide to stay on the common channel with probability p. If the node decides to stay on the common channel, it marks the channel as "HIGH" in its PCL. (For details of MMAC, refer to Chapter 3). When a node stays on the common channel, it can receive synchronization messages from all of its neighbors, and synchronize to the fastest node. With this scheme, eventually all nodes can synchronize their clocks with each other. The convergence time will depend on the probability p, which is a trade-off with channel utilization.

To evaluate the performance of MTSF on MMAC, we have implemented the protocols in ns-2 simulator. For the physical layer, we use IEEE 802.11b. 802.11b has 11Mbps of maximum rate, and has 3 non-overlapping channels. The transmission range of each node is approximately 250 m. For MMAC, the beacon interval is set to 100 ms, and the duration of channel negotiation window is 20 ms. For the routing protocol, AODV is used. Each node is assigned a *clock drift*, the speed at which the clock runs. For example, if the clock drift is 0.01, then it means that the clock is running 1% faster than the "real" time. If clock drift is zero, then the clock is perfectly accurate. The clock drift is randomly picked from the range [-0.0001, 0.0001]. All clocks are synchronized when the simulation starts, and start drifting according to the assigned clock rate. (It is possible that if the clocks not synchronized initially, transmitted beacons may not be received by the neighbors because they are not on the common channel. To address this problem, each node may need to periodically stay on the common channel for the whole beacon interval, so that it can receive beacons from its neighbors. Here we assume that clocks are initially synchronized.)

In MTSF, for the probability of a leaf node sending a beacon message, we use 10%. Also, the probability of a node staying on the common channel for the whole beacon interval is set to 10%. For traffic, we use CBR (Constant Bit Rate) traffic over UDP. Finally, each simulation is run for 300 seconds.

In the graphs, "Perfect" refers to the case where all nodes have accurate clocks. This case is included for the purpose of comparison. For the case of "Drift" and "MTSF", clocks are assigned different rates as described above, so that the clocks drift away from each other. In the case of "Drift", no synchronization protocol is run, and in the case of "MTSF", the MTSF protocol is run.

Figure 5.1 shows the aggregate throughput of the network over time. Without clock synchronization, the throughput starts degrading after some time, when the clock error becomes significant enough to affect channel negotiation. When MTSF is used, the throughput is similar to that of the case where clocks are perfectly accurate. Figure 5.2 shows the average throughput during simulation time. Although the throughput of "MTSF" is slightly lower than that of "PERFECT" due to the synchronization overhead, MTSF can help MMAC to achieve the benefit of using multiple channels at a small cost.

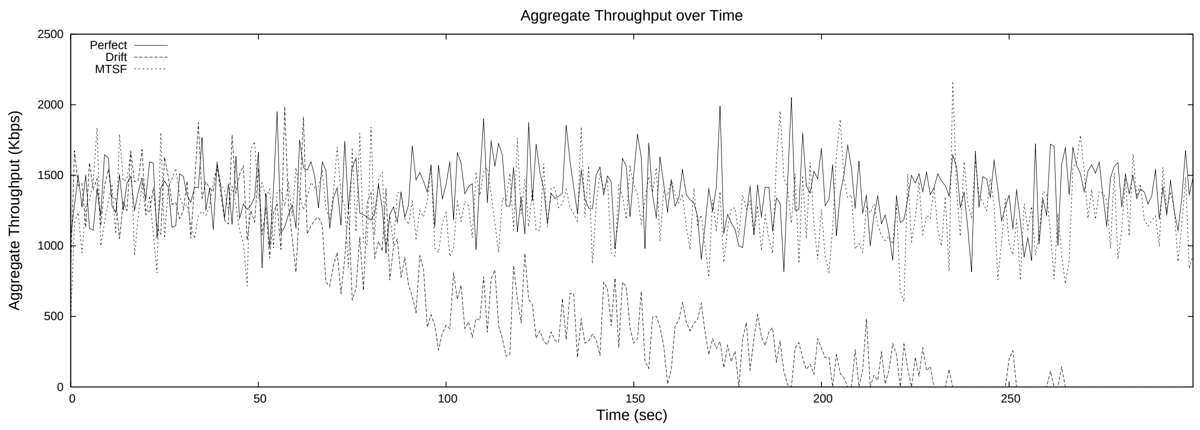


Figure 5.1: Simulation result: aggregate throughput over time.

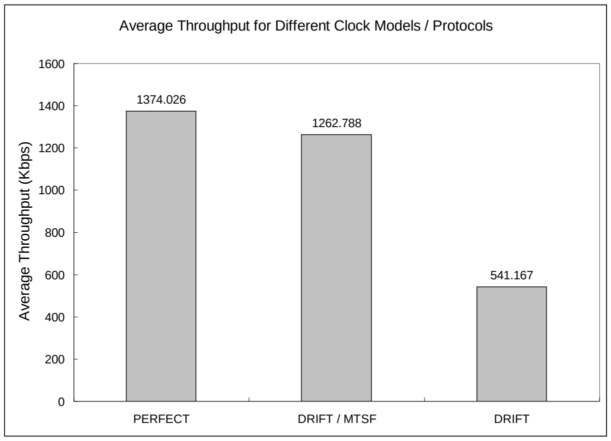


Figure 5.2: Simulation result: average throughput.

5.2 Integrating Routing with MMAC

Until now, we have used AODV as the routing protocol when running MMAC. AODV is one of the most widely used routing protocol in single channel multi-hop network. However, we observe that AODV may not utilize channel diversity well in a multi-channel network, especially when used with MMAC. In this section, we identify the problem with using AODV as the routing protocol on top of MMAC and propose a modification to AODV that can improve channel utilization. Since MMAC is designed for nodes with single interface, we assume that every node is eqiupped with a single interface.

5.2.1 Behavior of MMAC in Multi-Hop Flows

We first observe how MMAC chooses channels for a multi-hop flow. Since MMAC is a link layer protocol, it only considers choosing the "best" channel for each link. Consider the following scenario in Figure 5.3, with 4 nodes in a chain topology. In the figure, suppose node A establishes a route to D via B and C, and sends packets through the route. After some time, at the beginning of a beacon interval, node A will have packets for B, B will have packets for C, and C will have packets for D. Thus, nodes A, B, and C sends ATIM packets during the channel negotiation window. Since ATIM packets are sent after a random delay, the order of nodes sending ATIM packets is also random. Suppose in the ith beacon interval, node A sends an ATIM to node B first. They choose channel 2 as their data channel. After that, node B sends an ATIM packet to C. Since B has already chosen channel 2, node C also chooses channel 2. When node C sends an ATIM to D, D also chooses channel 2. So all the links in the path chooses the same channel, and channel diversity cannot be utilized within a flow. Suppose in the $i + 1$th beacon inveral, A sends an ATIM to B first, and they choose channel 2. After that, C sends an ATIM to D. Since C and D have not chosen a channel yet, they choose channel 3 (since channel 2 is used at node B). In this interval, nodes A and C can transmit their packets simultaneously on different channels, and node B has to buffer its packets until the next beacon interval. However, this selection of channels to not increase the flow throughput, because the link between B and C becomes a bottleneck. Ideally, the maximum flow throughput that this 3-hop flow can obtain is $c/2$, where c is the channel bit-rate (channel bit-rate is same for all channels). This throughput can be obtained with the schedule depicted in

Figure 5.4. However, with MMAC, this schedule cannot be obtained because if the link between B and C is assigned a channel, node A and D will also be assigned the same channel. Thus, the flow throughput results in $c/3$.

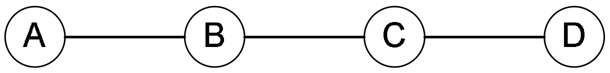


Figure 5.3: An example network scenario with 4 nodes.

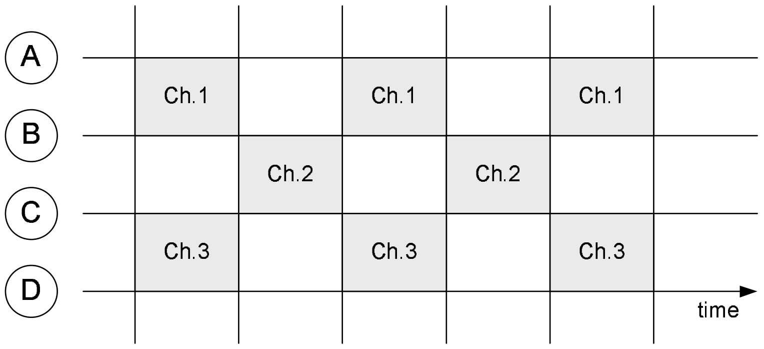


Figure 5.4: A link schedule that can achieve the maximum throughput in the scenario shown in Figure 5.3

Thus, we can conclude that MMAC cannot benefit from channel diversity within a flow. Figure 5.5 shows the result of a simulation we ran using the scenario in Figure 5.3. The throughput MMAC is approximately 80% of 802.11, reflecting the overhead of channel negotiation window.

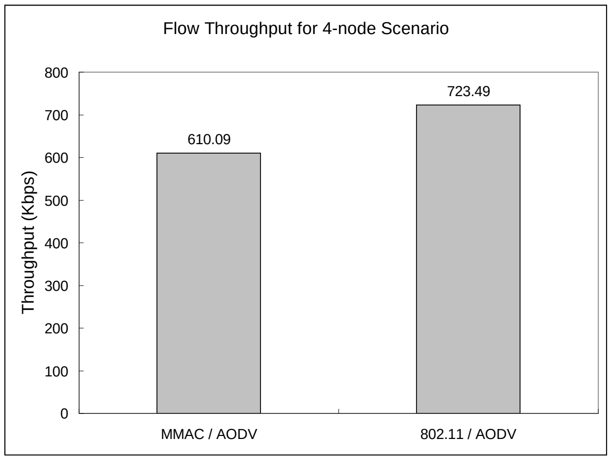


Figure 5.5: Flow throughput for 4-node scenario.

Now let us consider a scenario where there are multiple flows. In Figure 5.6, Node A has traffic to send to C and node D has traffic for F. When AODV is running as the routing protocol, both flows can take either node B or E as their relay node. Suppose both flows take node B as the intermediate node, as in Figure 5.7(a). In this case, MMAC cannot benefit from channel diversity and will assign the same channel for all links, because every link has node B as an endpoint. Once node B chooses a channel, all other nodes must follow that channel. Consider another case where node A selects B as its relay, and node D selects E as its relay, as in Figure 5.7(b). In this case, two flows can be assigned a different channel, thereby reducing contention and increasing throughput.

Due to the behavior of MMAC, it is beneficial that the routing protocol finds routes that do not overlap, if such routes exist. For single-channel networks, overlapping routes do not affect the performance, because nodes have to share the capacity of a single channel. (In Figure 5.6, between node B and E, only one node can transmit if they are using the same channel.) However, in multi-channel networks, overlapping routes limit the possibility of assigning different chanenls. Figure 5.8 shows the aggregate throughput of the scenarios in Figure 5.7.

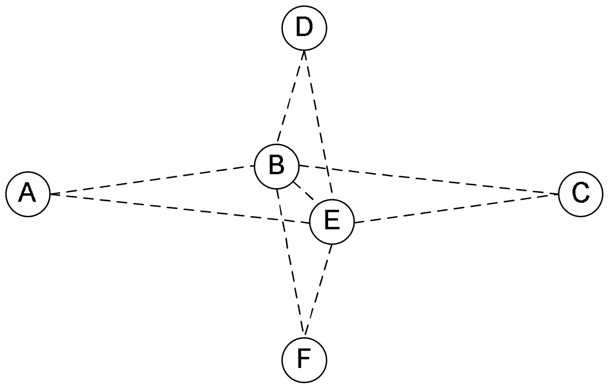
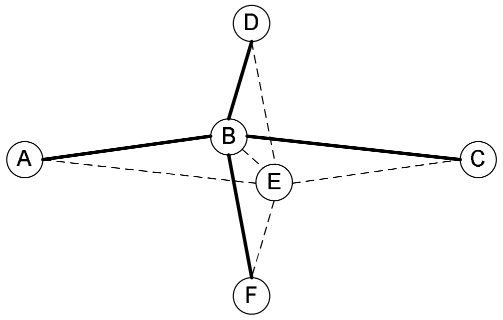


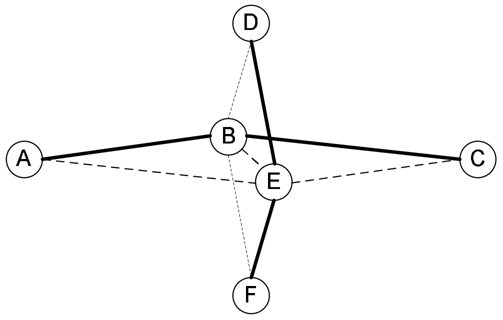
Figure 5.6: An example scenario with 6 nodes to illustrate the behavior of MMAC when multiple flows exist.

5.2.2 Extending AODV to Prefer Non-overlapping Routes

As we have seen previously, MMAC can benefit from non-overlapping routes in utilizing channel diversity. However, AODV does not take into account route overlaps when chooseing routes. Here, we make a modification to AODV to prefer non-overlapping routes, and study the impact on the network performance when implemented with MMAC as the MAC protocol. We call the modified protocol, AODV-E (AODV-Extension).



(a) Both flows select node B as their intermediate relay.



(b) Node A selects B as its relay, and D selects F as its relay.

Figure 5.7: Two possible cases of route selection in the scenario shown in Figure 5.6

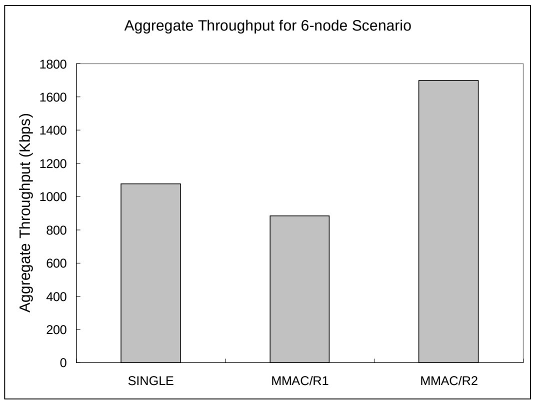


Figure 5.8: Aggregate throughput for the scenario shown in Figure 5.6. "SINGLE" refers to the case where IEEE 802.11 DCF is used, which can only use a single channel. "MMAC/R1" refers to the case where routes are chosen as in Figure 5.7(a). "MMAC/R2" refers to the case where routes are chosen as in Figure 5.7(b).

The basic idea of AODV-E is to include flow information in the Route Request (RREQ) packets, so that the destination node can select a route with minimum number of flows. Compared to the basic AODV, the following changes are made. (Note that in AODV-E, a "flow" is defined as a source-destination pair.)

- A route is maintained for each source-destination pair (or flow), instead of only the destination.
- RREQs include number of overlapping flows on the path, in addition to number of hops from the source.

- An intermediate node may forward RREQ multiple times, if the latter RREQ may lead to a route with a better metric than the former.
- Instead of selecting the route using the first arrived RREQ, the destination node waits for some interval so that it can receive multiple RREQs and choose the best one among them. In the simulations, we use 100ms for the interval.
- When the destination selects a route, it considers the number of overlapping flows as well as the number of hops from source to destination, as explained below.

We illustrate the route discovery process of AODV-E using the example scenario in Figure 5.6. Suppose node D has established a route to node F through node E, and is currently sending traffic over the route. Now node A wants to find a route to node C. To find node C, node A broadcasts a RREQ packet. In the RREQ packet, node A includes the number of flows that it is currently involved in, which is the number of route entries in the route table. (In the example scenario, since node A has no entry in the route table, it records zero in the RREQ.) The RREQ will be received by node B and node E. The intermediate nodes adds its number of route entries to the RREQ packet. Since node E has one entry for the route from D to F, E adds 1 to the RREQ packet. On the other hand, since node B does not have any entry, it adds 0 to the RREQ packet. When the intermediate nodes forward the RREQ packet, the destination node (node C in the example) will receive the RREQ packet. The content of RREQs received from node B and node E are shown in Figure 5.9.

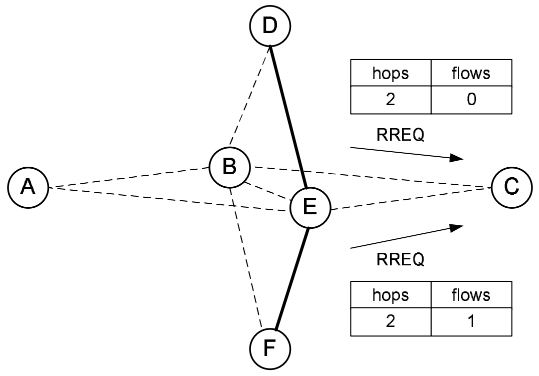


Figure 5.9: Illustration of route discovery process in AODV-E.

When the destination node receives the RREQ packets, it selects a route based on number of hops and number of overlapping routes. In the simulations, we preferred the number of overlapping routes to the number of hops. So a route with less number of overlapping routes are selected, and if there is a tie, a route with less number of hops is selected. The disadvantage of this stretagy is that a very long route may be selected if the route has the least number of overlapping routes. The stretagy can always be varied by tuning the priority of the two metrics: number of hops and number of overlapping routes. Once the route is selected, the destination node sends back a RREP packet to the source node, as in the original AODV.

5.2.3 Simulation Results

We have run simulations to compare the performance of MMAC/AODV and MMAC/AODV-E in scenarios with random topology. 50 nodes are placed in the simulation area of 600m $\times$ 600m. Within the simulation area, there is a inner square of size 500m $\times$ 500m. Among 50 nodes, 30 nodes are randomly placed within the inner area, and the other 20 nodes are placed in the outer area. When creating flows, the source and the destination nodes are chosen from the nodes in the outer area. Also, the source and its destination pair are chosen so that they are in the opposite side of the simulation area. The reason for doing this is to create multi-hop flows. When the source and the destination is within transmission range of each other, then both AODV and AODV-E will choose the single-hop route, so there will be no difference between the throughput of two routing protocols. One example of our simulated scenario is in Figure 5.10.

Similar to the simulations in the previous section, we used CBR traffic for the flows. Each flow generates 1000 packets a second, and the size of a packet is 512 bytes. The channel bit rate is 11Mbps. Under this simulation model, we have compared the performance of three schemes: IEEE 802.11 DCF with AODV, MMAC with AODV, and MMAC with AODV-E. The DCF protocol can only make use of a single channel, so it is included as a baseline for comparing AODV and AODV-E. When running MMAC, the drifting clock model described in the previous section is used, and MTSF is used to synchronize the clocks. Each simulation was run for 50 seconds.

In Figure 5.11 and 5.12, we have compared the protocols varying number of flows. For this simulation, we have used 12 channels. The bit-rate of each channel is 11Mbps. In Figure 5.12,

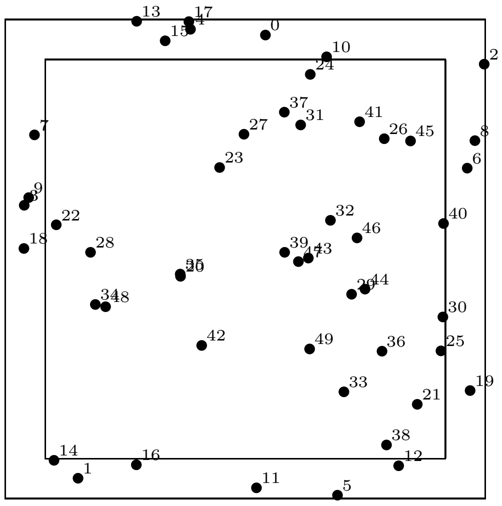
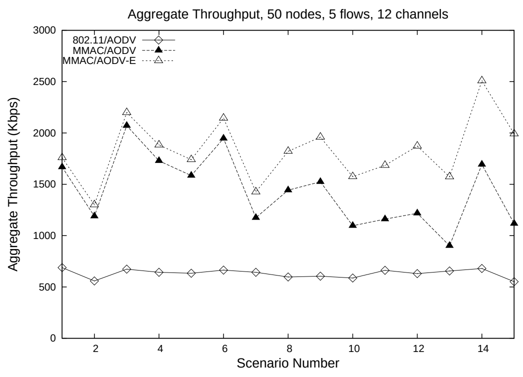


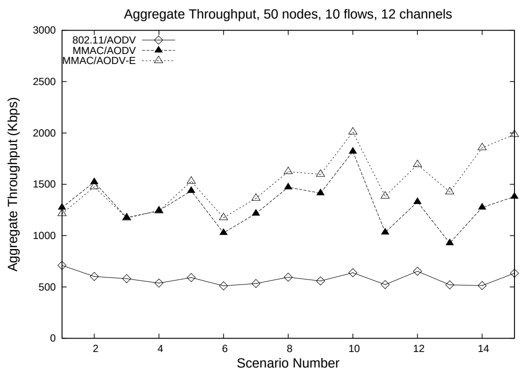
Figure 5.10: An example topology used for simulation. Nodes in the outer area can become sources or destinations.

throughput of the protocols were normalized to that of MMAC/AODV. With 5 flows, we could see up to 70% improvement for MMAC/AODV-E, compared to MMAC/AODV. This improvement is due to the fact that AODV-E tries to find non-overlapping routes, and MMAC can benefit from non-overlapping routes by assigning different channels to the flows. In some scenarios, the throughput of AODV and AODV-E are not different. In these scenarios, AODV chose non-overlapping routes by chance, although AODV does not consider if routes are overlapping or not. When the number of flows is increased to 10, the throughput gap is reduced, compared to the result with 5 flows. This is because there are not enough relay nodes to establish non-overlapping routes for all 10 flows. As mentioned earlier, we have placed 30 relay nodes in the inner region of the simulation area. The length of the routes were mostly 4 or 5, meaning that 3 or 4 relay nodes are used for each flow. Thus, overlapping routes are established with AODV-E, reducing the gap between AODV and AODV-E. AODV-E can benefit from dense environment, where the chance of establishing non-overlapping routes is higher.

In Figure 5.13 and 5.14, we have compared the protocols varying number of channels, while fixing the number of flows to 5. For this simulation, we have used 3 channels and 12 channels. In Figure 5.14, the throughput of the protocols were normalized to the throughput of MMAC/AODV. (Figure 5.11(a) and Figure 5.13(b) are identical, and also Figure 5.12(a) and Figure 5.14(b) are identical.) The result indicates that AODV-E can benefit more when there are large number of

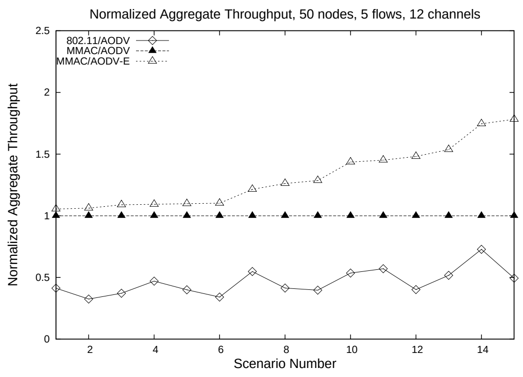


(a) 5 flows

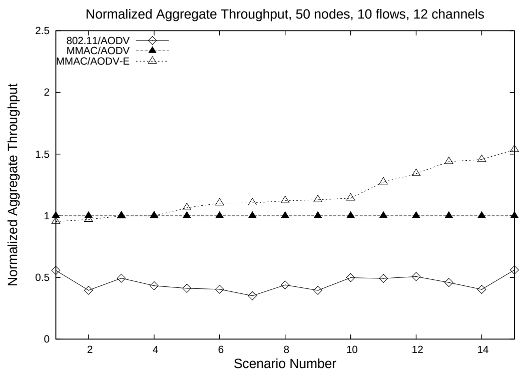


(b) 10 flows

Figure 5.11: Simulation result varying number of flows: aggregate throughput with 50 nodes and 12 channels.



(a) 5 flows



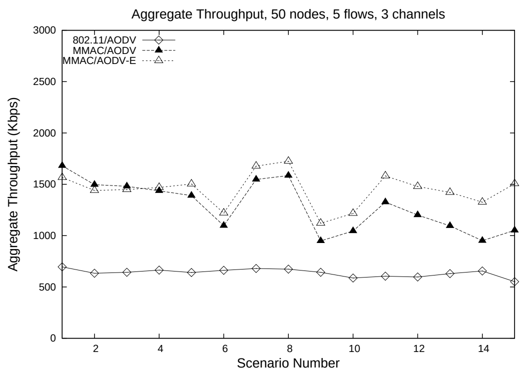
(b) 10 flows

Figure 5.12: Simulation result varying number of flows: normalized aggregate throughput with 50 nodes and 12 channels.

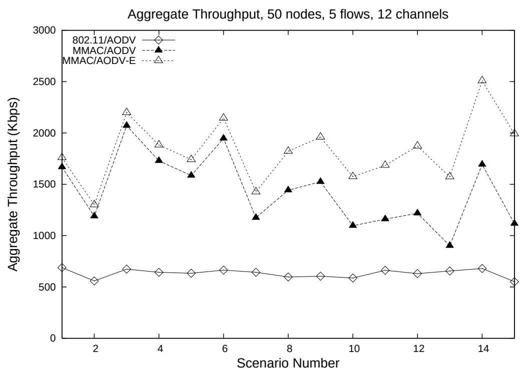
channels. This is intuitive, because with 3 channels, even though AODV-E finds non-overlapping routes, some routes need to be assigned the same channel because there are only small number of channels. From the results, we can conclude that AODV-E can help MMAC in utilizing multiple channels by finding non-overlapping routes, and the amount of benefit is larger when the network is dense and more number of channels are available.

5.3 Summary

In this chapter, we have integrated MMAC with MTSF, the two protocols proposed in the previous chapter. Also, we introduced an extension to AODV that can perform better than original AODV

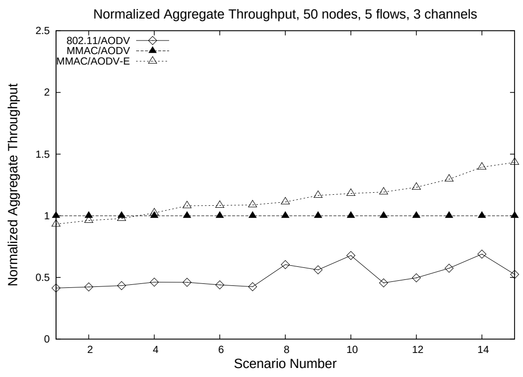


(a) 3 channels

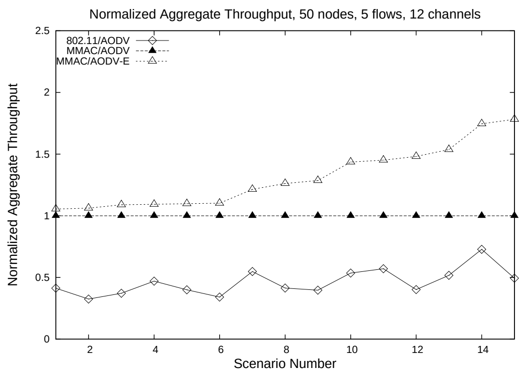


(b) 12 channels

Figure 5.13: Simulation result varying number of channels: aggregate throughput with 50 nodes and 5 flows.



(a) 3 channels



(b) 12 channels

Figure 5.14: Simulation result varying number of channels: normalized aggregate throughput with 50 nodes and 5 flows.

when used with MMAC. Clock synchronization is a crucial part in making MMAC to work in a multi-channel multi-hop environment. In Chapter 3, the clock synchronization was assumed to exist. However, we have shown in this chapter that MTSF can maintain clock accuracy to the level that MMAC can work well, at a small cost of exchanging synchronization messages. Especially, with MMAC, the overhead can be further reduced by piggybacking timestamps in ATIM packets. In Chapter 3, AODV was used as the routing protocol in evaluation. AODV tries to find a route with short hop-distance, but does not particularly prefer non-overlapping routes. However, we have observed that due to the behavior of MMAC, finding non-overlapping routes can be useful for MMAC in assigning channels. Thus, we proposed an extension to AODV, called AODV-E, that tries to find non-overlapping routes. Through simulations we have shown that AODV-E can produce better performance with MMAC in multi-channel multi-hop environment, compared to the original AODV.

Chapter 6

Multi-Channel Routing Protocols for Ad Hoc Networks

While multi-channel MAC protocols combine channel assignment with medium access control, the channel assignment can also be done at the network layer, combined with the routing protocol. In this chapter, we develop multi-channel routing protocols for multi-hop networks. In particular, we assume that the MAC layer is IEEE 802.11 DCF. The benefit of having channel assignment functionality at the network layer is that it can be easily deployed by only modifying software at routers.

There are several routing protocols for multi-channel networks [15,51], but they require multiple transceivers at each node. Here we consider the environment where each node is equipped with a single transceiver.

In the rest of the chapter, we first review the existing work in the area of multi-channel routing. Then, we discuss issues regarding routing in multi-channel environment. Based on the discussion, we develop a multi-channel routing protocol which performs channel assignment at the time of route establishment. We provide results to show that multi-channel routing protocols can utilize multiple channels with a single transceiver at each node.

6.1 Related Work

In this section, we review existing routing and channel assignment protocols for multi-channel networks.

Shacham and King [52] proposed routing protocols for multi-channel networks. The paper proposes different schemes for different scenarios. In the first scenario, every node has a single transceiver. In this case, every node broadcasts its routing information in all channels. After gathering the route information, a sender switches to the destination's channel and start transmitting packets. This scheme assumes that every node has at least one neighbor for every channel. This may be true for dense networks, but is often not true in a sparse networks. In the second scenario, there are nodes with multiple radios. Then they can act as relays and forward packets from one channel to another. This scheme requires enough number of nodes with multiple radios to establish routes between any pair of nodes in the network.

Raniwala et al. [51] proposed a centralized channel assignment algorithm assuming that each node has two transceivers. With N channels available, this protocol assigns these channels to nodes so that the network is not partitioned and bandwidth can be efficiently allocated. Adya et al. [53] proposed a link-layer protocol that manages underlying multiple interfaces. The protocol attempts to select channels so that the overall channel utilization is optimized.

Chandra et al. [54] propose a software layer which enable a single wireless card to connect to multiple networks. Also, the paper proposes algorithm for switching between networks and protocol for buffer management. Finally, Draves et al. [15] proposes a metric for multi-channel networks combining expected transmission time and channel diversity.

6.2 Routing with Multiple Channels

In this section, we discuss the issues in designing a routing protocol for multi-channel multi-hop networks. We investigate different approaches, and identify the benefits and problems of each approach. The routing protocol proposed in the next section is a result of this discussion.

As mentioned before, our goal is to design a routing protocol that can utilize multiple channels without modifying the MAC protocol. Also, we assume that each node has only a single transceiver, and thus capable of listening to only one channel at a time. Since traditional MAC protocols such as IEEE 802.11 DCF are designed for a single channel network, the routing protocol must perform channel assignment as well as route discovery and maintenance.

Consider the scenario illustrated in Figure 6.1, where five nodes form a multi-hop network. If two neighboring nodes want to communicate, they must agree on a common channel on which they will communicate.

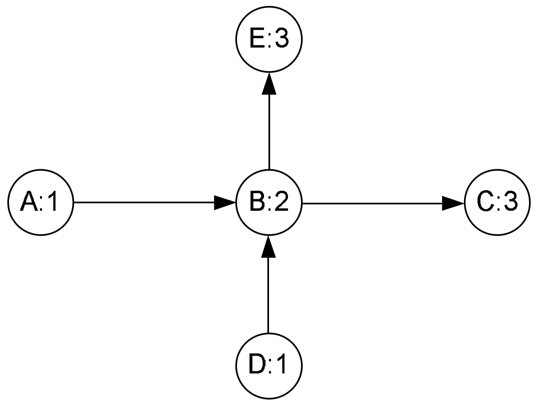


Figure 6.1: An example network scenario. The label in each node indicates the node id and the channel it is listening on.

We define a *flow* to be a connection between a source-destination pair. Suppose node A wants to initiate a flow with node C. Then node A must first obtain a route to node C, if it does not already have one. For a single-channel network, establishing a route means to obtain a path where each node in the path knows the next hop towards the destination. However, in a multi-channel network, each node in a path needs to know on which channel it should transmit packets, so that it can reach the next hop and eventually the destination.

There are multiple ways to assign channels, and we investigate two approaches: assigning channels to nodes, and assigning channels to flows.

6.2.1 Two approaches for channel assignment

The first approach in channel assignment is to assign channels to nodes regardless of the traffic patterns. In this approach, nodes do not switch channels to participate in a flow, but each node in the path knows the next hop node and the channel it is listening on.

Let us consider the scenario in Figure 6.1 again. Suppose node A is initially listening on channel 1, node B on channel 2, and node C on channel 3. For node A to send packets to C, A needs to know that B is the next hop node and it is listening on channel 2, and B needs to know that C is the next hop node and it is listening on channel 3. After establishing a route, nodes can send packets by switching to the channel of the receiver, whenever they have packets to send.

The benefit of this approach is that route establishment is decoupled from channel assignment. It makes both routing and channel assignment algorithm simple. For example, channels can be randomly assigned to nodes, and each node can broadcast its route information on all channels. With these simple mechanisms, all nodes will eventually obtain routes to every other node in the network.

However, we argue that assigning channels to nodes can result in significant performance degradation. It is due to the *deafness problem*, in which two nodes cannot communicate because they are listening on different channels. Deafness problem is first identified in the context of directional antennas [55], where nodes cannot communicate with each other because their antennas are beamforming in other directions. Suppose in Figure 6.1, there is a flow from A to C and another flow from D to E. Node A and D are on channel 1, node B is on channel 2, and node C and E are on channel 3. When B receives a packet from A, it switches to channel 3 to forward the packet to C. When it sends the packet, it returns to channel 2. While node B is on channel 3, node D has a packet to send to B, so it transmits the packet on channel 2. But since B cannot receive the packet, the packet will be lost.

In this case, MAC protocols such as IEEE 802.11 DCF will wait for a random delay and retransmit the packet. This results in unnecessarily increased delay. Moreover, since D does not know when B is returning to channel 2, the retransmission may fail again. After several retransmissions, DCF drops the packet and notifies the routing protocol that the link has broken. Then the routing protocol assumes that the route has been broken, and take actions accordingly.

The second approach in channel assignment is to assign channels to flows instead of nodes. This means that whenever a route is established, all nodes in the route are assigned to a common channel. In this case, the channel assignment needs to be coupled with route establishment.

This approach works well with on-demand routing protocols, since channels are assigned at the time of route discovery. But it is difficult to maintain routes proactively. Note that for the first approach where channels are assigned to nodes, both proactive and reactive routing schemes can work.

The benefit of this approach is that once the route has been established, nodes in the path do not need to know which channel the next hop is listening on, nor do they need to switch channels to transmit packets to the receiver. The deafness problem can be avoided, because nodes are not switching channels. The downside of this approach is that the routing protocol becomes complicated, as explained in the next subsection. We choose this approach in this chapter.

6.2.2 Assigning channels to flows

In a very dense network, it might be possible to find node-disjoint routes for all flows, but for a mid-sized network with many flows, intersecting routes can be a common case. It is not easy to assign channels if multiple flows share a node as an intermediate or end node. For example, consider the scenario in Figure 6.2. Suppose there are two flows, one from A to E and another from F to I. These two flows intersect at node C. To assign a common channel to all nodes in a flow, the same channel has to be assigned to these two flows, and the benefit of having multiple channels is nullified. This problem becomes worse as the number of flows increases, because the chance of two flows intersecting at a node increases. One example of this situation is illustrated in Figure 6.3. Also, suppose a new route must intersect with two existing routes that are node-disjoint and operating on different channels. This route cannot be established without forcing the already existing flow to switch channels. It is certainly undesirable if the routes cannot be established even though there is a path between the source-destination pair.

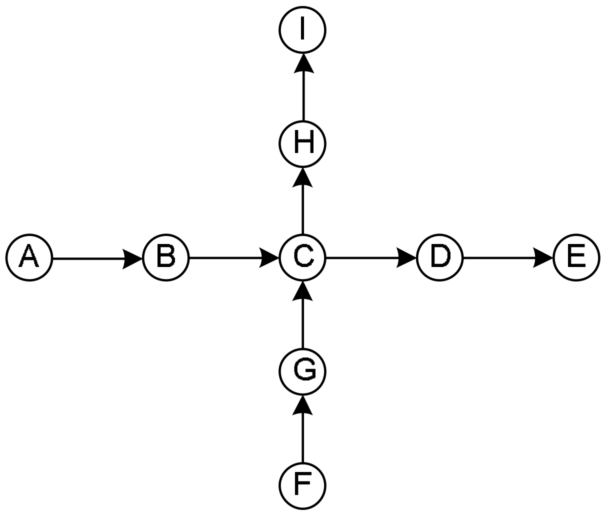


Figure 6.2: An example network scenario. Two flows A-E and F-I are intersecting at node C.

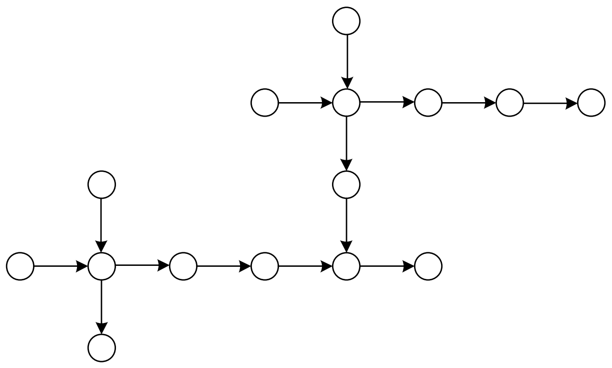


Figure 6.3: An example network scenario. If channel switching is not allowed, all flows must be assigned the same channel.

To improve performance and route discovery success ratio, we need to allow some nodes to switch channels. Suppose in Figure 6.2, if node C can switch between two different channels, we can benefit from assigning different channels to these nodes, because transmissions such as A-B and C-H can take place simultaneously. However, allowing nodes to switch channels can lead to deafness problem, as we have explained in the previous subsection. To avoid the deafness problem, we allow nodes to switch channels with the following constraints.

- Two consecutive nodes in a path cannot switch channels. One of them has to be fixed on a channel. In our protocol, we take a conservative approach and allow only one switching node per path.
- A node must notify its neighbors in a path whenever it switches channels.
- A node can only switch between a small number of channels (such as two), although many more channels are available.
- Nodes may not switch channels too frequently, such as per-packet basis.

The first two constraints are necessary to avoid the deafness problem, and the last two are for performance reasons. Since there is a channel switching delay of 80 μs, it is not desirable to have one node switch between too many channels, or switch channels frequently.

Returning to Figure 6.2, we can assign channel 1 for the route between A and E, and channel 2 for the route between F and I, and have node C switch between channel 1 and 2. Whenever C switches channels, C can broadcast messages on channel 1 and 2, so that the neighbors of node C knows which channel C is currently on.

Even with these constraints, allowing some nodes to switch channels gives much more freedom to the routing protocol in selecting routes and assigning channels, and improves the route discovery success ratio. However, still there are cases where the route discovery may fail because of the constraints. For example, suppose there are three nodes A, B, and C that are already assigned channels 1, 2, and 3, respectively. If the only available route is to go through these three nodes, no channel can be assigned to the route without creating two switching nodes. In this case, forcing other routes to switch channels is inevitable. In the worst case, all flows may fall back to sharing a common channel, but still the performance should not go below that of a single channel protocol.

6.2.3 Channel Load Balancing

Until now, we have discussed different approaches of assigning channels. We choose the approach where channels are assigned to flows, and channel switching is allowed to improve the freedom of choice in route selection. The route discovery returns multiple routes to choose from, and the protocol needs to select a route and also the operating channel for the route.

For single-channel routing protocols, routes are selected based on many different metrics such as hop distance, signal strength, degree of stability, and expected transmission time. For a multi-channel routing protocol, balancing load between available channels is another important goal.

Consider the scenario in Figure 6.4, where all pairs of nodes are at a single hop. Three flows can be established on node-disjoint routes. If there is no knowledge of traffic load on each channel, the protocol can randomly assign a channel for each flow. However, this may result in all three flows being assigned with the same channel. To be able to assign different channels to flows that are close to each other, nodes need to collect information on channel load. One possible solution is to use HELLO messages, where each node periodically transmits HELLO messages on all channels, in a round robin manner. Sending HELLO messages too frequently may be too expensive. Also, since a node has to switch channels when sending HELLO messages, the node becomes temporarily deaf. On the other hand, if the period of sending HELLO messages is too long, the collected information may be stale. Thus, there is a trade off between overhead and the accuracy in collected channel load information.

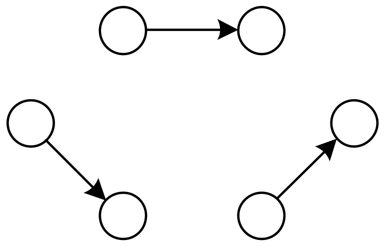


Figure 6.4: An example network scenario with three node-disjoint flows. Random channel assignment may result in the same channel assigned for all flows.

The discussion presented in this section leads to the proposed protocol, called Multi-Channel Routing Protocol (MCRP), which is explained in more detail in the next section.

6.3 Multi-Channel Routing Protocol (MCRP) [3]

In this section, we present the proposed Multi-Channel Routing Protocol (MCRP). MCRP is an on-demand routing protocol, and has many similarities with AODV [18]. However, MCRP benefits from multiple channels without requiring change in the underlying MAC protocol. It improves network performance by trying to allocate different channels to different flows, thus allowing simultaneous transmissions in a region. In the worst case, the performance of MCRP is comparable to AODV which is an on-demand single channel protocol. The MCRP protocol attempts to ensure that a source will establish a route to the destination, if it could find a route in a single channel network with the same topology. MCRP assigns a common channel to all nodes in a flow, based on the discussion in the previous section. MCRP allows nodes to switch channels, but does not allow two consecutive nodes in a flow to switch channels, which causes deafness problem as also explained in the previous section.

Now we describe the protocol in detail. Since the MCRP shares many common features with the well-known AODV [18], we focus on the unique parts of MCRP, and omit the parts that are same as in AODV.

6.3.1 Node States

In MCRP, each node is in one of the four *node states* explained below. Suppose we have k channels. When a node is turned on, it may choose to stay on any of the k channels. When the node does not have any traffic it is responsible for transmitting, it can freely switch to other channels. We say that this node is in *free state*, and we call this node a *free node*. Once the node becomes a part of a traffic path and is assigned a channel, the node becomes a *locked node*. It is possible that two flows with different channels can intersect at a node. In this case, the node where two flows intersect needs to switch between two channels. This node is called a *switching node*. As explained later, MCRP prohibits a node to switch between more than two channels, due to excessive channel switching overhead. Also, as mentioned before, MCRP does not allow two consecutive switching nodes in a flow. Thus, the neighbors of a switching node in a flow become *hard-locked nodes*, and they must not be made switching nodes in order to prevent deafness problem. These four states are called *feasible states*, and a node must be in one of the four states. For example, consider the scenario with two flows in Figure 6.2. Flow 1 travels along the path A-B-C-D-E, and flow 2 uses the path F-G-C-H-I. Suppose the protocol assigns channel 1 to flow 1 and channel 2 to flow 2. Then node C becomes a switching node. Nodes A, C, F and I are locked nodes, because they are involved in flows. Nodes G, B, D, H are hard-locked nodes, because they are neighbors of C in a flow and must not become switching nodes. Nodes that are not involved in any flow are free nodes. These node states affect route selection and channel assignment. The four node states are summarized in the following.

- free: The node does not have any flows and may freely switch to other channels.
- locked: The node has a flow on a certain channel.
- switching: The node is involved in multiple flows on different channels.
- hard-locked: The node has a flow on a certain channel, and it cannot be made a switching node.

6.3.2 Route Discovery

When a node has packets to send, it initiates a route discovery by broadcasting a Route Request (RREQ) packet. In a multi-channel network, nodes may stay on different channels. Thus, the RREQ packet must be broadcasted on all channels, as opposed to one channel in a single channel protocol. Suppose node S wants to find node D in a network with 3 channels. Also, suppose that node S was initially on channel 1. Then, S broadcasts RREQ in all channels, in a round robin manner. More specifically, S broadcasts RREQ on channel 1, switches to channel 2, broadcasts RREQ on channel 2, and so on. After going through all channels, the node returns to the original channel. Now when an intermediate node receives RREQ, it forwards the packet on all channels. The intermediate nodes also return to their original channel after forwarding the packet. Since RREQ packets are forwarded on all channels, node D can receive the packet regardless of which channel it is on. Also, as the RREQ is forwarded, the nodes set up a *reverse path* to node S, as in AODV [18].

The number of RREQ packets transmitted is at most kn, where k is the number of channels, and n is the number of nodes in the network. For a single channel routing protocol which uses flooding for route discovery, the overhead is at most n. Thus, although the route discovery overhead of MCRP seems to be large, the overhead per channel is the same as that of a single channel protocol. Note that during the route request process, deafness problem may occur temporarily because nodes are switching channels.

The route entries of MCRP are as shown in Figure 6.5. In the route entry, all fields except *channel* and *active* fields are also in AODV. The *channel* field indicates the channel that the next hop node is on. So the node has to transmit a packet on the channel indicated in the *channel* field to reach the next hop. The *active* field is only relevant when the next hop node is a switching node (otherwise this field will have value 1). It indicates whether the next hop node is on the channel specified in the *channel* field. If this field has value 0, all packets that use this route must be buffered until the field becomes 1. How to manage the "active" field is explained later. (The "dest" field indicates the destination node id, the "seqno" field indicates the destination sequence number, the "hops" field indicates the number of hops to the destination, the "nexthop" field indicates the id of the next hop node towards the destination, the "expire" field indicates the time when this route expires, and the "flags" field indicate whether this route is working or has failed.)

dest	*seqno*	*hops*	*nexthop*
channel	*active*	*expire*	*flags*

Figure 6.5: A route entry in route table of MCRP.

When a node sets up a reverse path to the source node, it needs to know which channel the next hop node is listening on. In order to provide this information, a node includes its operating channel (the channel which the node was on before broadcasting the packet) in the RREQ packet. This channel information is used by the nodes that received the RREQ packet to establish reverse path to the source node.

Finally, RREQ reaches the destination node D. Then, node D selects the channel which is to be used for the flow. The channel selection mechanism is explained later. After selecting a channel, node D sends a route reply (RREP) to node S using the reverse path.

As the RREP travels from node D to S, the nodes in the path switch channels and also node states according to the following. Suppose the selected channel is channel 1.

- free node: The node becomes a locked node and switches channel to channel 1.
- locked node: If the node is locked on channel 1, nothing changes. If the node is locked on another channel, it becomes a switching node between channel 1 and the original channel that this node was on.
- switching node: If channel 1 is one of the channels it is switching between, then nothing changes. Otherwise, the node drops the RREP packet. This is because MCRP does not allow a switching node to switch between three or more channels, as mentioned before.
- hard-locked node: If the node is locked on channel 1, nothing changes. Otherwise, the node drops the RREP packet, because hard-locked nodes are not allowed to become switching nodes.

Note that the RREP packet is dropped if the selected channel makes the node enter an infeasible state. However, if there are large number of flows in the network, a flow might only finds paths that requires one of intermediate nodes to go into an infeasible state. If all these routes are dropped, the source node may not be able to establish a route to the destination. To avoid this, we use the "force" mechanism which forces other routes to change, as will be explained later.

When a route is no longer used and thus expires, it may bring changes to the node state. The node state changes according to the following rules.

- Regardless of the current state, a node becomes a free node if no active route remains.
- If a switching node no longer uses two channels and only sends and receives packets on a single channel for a certain duration of time, then the switching node becomes a locked node. (In the simulations, we use 3 seconds for the duration.)
- If a hard-locked node does not receive JOIN or LEAVE messages for a certain duration of time, it becomes a locked node. (In the simulations, we use 500 ms for the duration.)

An example of the route discovery process is illustrated in Figure 6.6. In the figure, there is a flow from X to Y on channel 2, and also a flow from M to N on channel 3. Suppose node S wants to find a route to node D. S starts a route discovery by broadcasting RREQ packets on all channels. After broadcasting on all channels, it returns to channel 1, where it was originally. When the RREQ is sent on channel 2, node A receives it and forwards it on all channels. Also, since node S includes its original channel in the RREQ, A knows that S is on channel 1 and sets up a reverse path to S. Node B does the same thing and finally node D receives the packet. When the source node and the intermediate nodes forward an RREQ, the nodes do not switch channels for a certain amount of time, so that it can receive RREPs on the current channel. (In the simulations, we use 100 ms for the interval.)

On receiving the RREQ pcaket, D sends back RREP on the reverse path to S. Since each node knows which channel the receiver is on, it sends the packet on the receiver's channel. In this case, D sends on channel 3, B on channel 2, and A on channel 1.

When the RREP packet reaches node S, the route is established and the data packets are transmitted on the selected channel. After a route has been found, if a route is not used for a certain duration of time (6 seconds in our simulations), the route expires and is deleted from the route table. When a data packet uses a route, the route is refreshed. In MCRP, both forward and

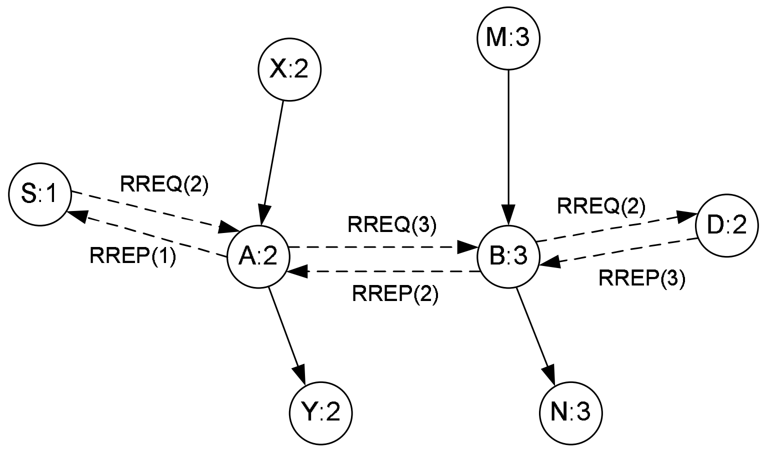


Figure 6.6: A network scenario to illustrate route discovery process. The label “A:2” means that node A is on channel 2.

reverse paths are refreshed. For example, in Figure 6.6, if node A receives a packet destined for D from S and forwards the packet to B, node A refreshes the route to node D (the forward path) and also the route to node S (the reverse path).

6.3.3 Channel Selection

When a destination node receives a RREQ packet, it needs to select a channel for the route. The channel selection algorithm must achieve two goals. First, no node in the path should go into an infeasible state. The channels that achieve this are called *feasible channels*. Second, among the feasible channels, the channel with the lowest load should be selected for the purpose of channel load balancing. For the first goal, a *channel table* is used, and for the second goal, a *flow table* is used (as described below). Both channel table and the flow table are carried in the RREQ packet to the destination.

Channel Table

The channel table records the state of nodes in the path from source to destination. As the RREQ packet is forwarded, each node updates the channel table included in the RREQ packet. Once the RREQ reaches the destination, the destination node selects the channel using the information provided from the channel table. The channel table contains a field for each channel as described in Figure 6.7.

ch_1	ch_2	...	ch_k

Figure 6.7: The channel table included in the RREQ packet.

Initially, all fields are set to 0. The rules for updating the channel table depend on the node state. The rules are as follows.

- free node: No changes made to the channel table.
- locked node: If the locked channel is channel i, increment ch_i by one.
- switching node: if the two channels are channel i and j, increment ch_i and ch_j by one.
- hard-locked node: If the locked channel is channel i, increment ch_i by two.

To illustrate the use of channel table, we use Figure 6.6 again. In the figure, there are two flows: one from X to Y, and the other from M to N. Because of the flows, nodes X, A and Y are locked on channel 2, and nodes M, B and N are locked on channel 3. Nodes S and D are free nodes. When S sends an RREQ, it sends the channel table with zeros in all fields. If there are three channels, the channel table will be (0, 0, 0). When A forwards the RREQ, it adds 1 to ch_2 and the channel table becomes (0, 1, 0). Similarly, B updates the channel table to be (0, 1, 1). When D receives the RREQ, since it is a free node, the final channel table is (0, 1, 1). Using this channel table, node D chooses one channel according to the scheme explained later.

Flow Table

When there are multiple candidates for a selected channel, the channel should be chosen which makes the load balanced among channels. Since the protocol assigns the same channel for all nodes in a flow, the channel condition near the intermediate nodes must be considered as well as the channel condition near the destination, when the destination selects a channel. More specifically, the channel should be chosen which maximizes the throughput at the *bottleneck node*. Suppose that metric $x_c(i)$ denotes the interference level of channel c around node i in the path from S to D. Then the *path interference level* I_{SD} is:

$I_{SD} = \min_c(\max_i(x_c(i), i \in P_{SD}), c \in C)$

where P_{SD} is the path from node S to D and C is the set of candidate channels. The selected channel is the one which has the minimum I_{SD}.

In our protocol, we use the *number of flows in the neighborhood* as the load metric. For each node, we count for each channel the flows that are going through itself and its neighbors. To collect the information from the neighbors we use HELLO mechanism. Each node transmits HELLO messages periodically, on all channels. Since a node can only transmit on one channel, it sends the packet in a round robin manner, as the RREQ packet. Again, this switching of channels may cause deafness problems, but the period of sending HELLO messages is long so that the problem does not affect the performance significantly.

In the HELLO packet, the node includes what channels it is on, and the number of flows in the channel. Using the HELLO messages and the flow state of itself, each node builds up its own *flow table*, as shown in Figure 6.8. The flow table records for each channel the number of nodes in the neighborhood that are currently using the channel (including the node itself). We denote $F_c(i)$ as the number of flows in channel c around node i.

The flow table, which is included in the RREQ packet looks like Figure 6.8.

F_1	F_2	...	F_k

Figure 6.8: The flow table included in the RREQ packet.

In the flow table, F_c means the number of flows for channel c. As the RREQ packet is passed, node i updates the flow table according to the following rule.

- For each c, if $F_c(i) > F_c$, then update F_c to be $F_c(i)$.

When the RREQ arrives at the destination, the destination node knows the interference level of each channel in the path.

We use Figure 6.6 once more to explain the flow table. Using the HELLO messages, each node builds up the flow table. Suppose we have three channels. Then after the two routes are established, A's flow table will be (0,3,1), because two active neighbors and itself are on channel 2, and one active neighbor is on channel 3. Node S's flow table will be (0,1,0), and node B's table will be (0,1,3).

In the following, the flow table indicates the flow table in the RREQ packet, not in a node. When S sends an RREQ, it updates the flow table to (0,1,0). When A receives it, it updates the flow table in RREQ to (0,3,1). Now when the RREQ comes to B, B updates the flow table to (0,3,3), following the rule above. Finally, D updates it to (0,3,3) which is the final flow table used in the channel selection.

Channel Selection Algorithm

When the destination node receives a RREQ packet, it needs to select a channel for the flow and sends back the RREP packet. The channel selection is performed using the channel table and the flow table included in the RREQ packet.

First, the destination node tests to see if the route is feasible, meaning that it does not create any two consecutive switching nodes in a flow or assign more than two channels for a node. The test is done using the channel table. The route is determined to be infeasible if:

- Multiple channels have values greater or equal to 2.
- More than two channels have values greater or equal to 1.

If any of these two conditions are met, then the route is considered infeasible. Then the route is either dropped, or is still used if the protocol uses "force mechanism" explained later.

If the route is considered feasible, then the destination node chooses the channel considering the flow table, according to the following rules.

- If a channel has a value greater or equal to 2, this channel has to be selected.
- If two channels have a value 1, then one of these two channels which has the minimum interference level is selected.
- If only one channel has a value 1 and all other channels have 0's, then among all channels, the one with the minimum interference level is selected.

Returning to the example in Figure 6.6, node D has the final channel table of (0,1,1), and the final flow table of (0,3,3). According to the above rules, this route is feasible, because only two channels have 1 as their values. Also, either channel 2 or channel 3 has to be selected because they both have a value 1. Now since two channels have the same interference level of 3, then one is chosen at random.

Delayed Reply

In a single channel on-demand routing protocol such as AODV, the destination node replies to the first route request and disregards all others. This is because the first route request to reach the destination has presumably taken the route with the least delay. However, in MCRP, the destination waits for a certain duration of time so that it can receive multiple RREQ packets, before choosing a route. (In the simulations, we use 50 ms for the delay.) If some RREQs arrive after the destination chooses a route, they are discarded.

Also, in AODV, the intermediate nodes only forward the route request once, due to the same reason that the first route request is taking the path of least delay. However, MCRP allows the intermediate nodes to forward the route requests, it it has a better metric than the first one.

6.3.4 Packet Forwarding and Channel Switching

Once a route has been established through the route discovery process, the source node can begin sending data packets to the next hop node in the path. Since MCRP assigns a common channel to all nodes in a flow, nodes can send packets on the selected channel without switching to another channel. However, communicating with a switching node is more complicated, since it switches channels from time to time to support multiple flows in different channels. A node may fail to send packets to a switching node, if the switching node is listening on another channel. Furthermore, a node may falsely think the route is broken after several tries, if the switching node stays on another channel for a long time, causing significant performance loss. So the neighbors of a switching node must know whether the switching node is available on the channel they are listening on.

To achieve this, a switching node uses LEAVE/JOIN messages to inform its neighbors of its channel status. Suppose a switching node B switches channels from 1 to 2. Before switching channels, B sends a LEAVE message on channel 1. The neighbors listening on channel 1 receive the LEAVE message and reset the 'active' flag in the route entries having node S as the next hop.

Then B switches its channel to channel 2. After switching channels, B sends a JOIN message on channel 2. The neighbors on channel 2 receive this message and set the 'active' flag in the route entries having node S as the next hop.

When a node has packets to send to a switching node which is currently listening on another channel, the routing protocol needs to buffer the packets until it receives a JOIN message from the switching node. For this reason, the routing protocol maintains a separate buffer for keeping the packets waiting for the switching node. When the node receives a JOIN message from the switching node, the packets in this buffer are sent with a higher priority then other packets.

Since LEAVE and JOIN messages are broadcast messages, receivers do not reply with an ACK. Loss of LEAVE and JOIN messages may lead to throughput degradation. Suppose node A is a fixed node and node B is switching node, and they are neighbors. If A does not receive a LEAVE mssage from node B, A can try sending packets to B while B is on another channel, which results in packet losses. If A does not receive a JOIN message from B, A might defer sending packets to B when B is on the same channel as A. Both cases lead to throughput degradation, but missing LEAVE messages has a more severe impact because A might think the link between A and B is broken. We do not address this problem in our work. A possible approach is to have a node send the data packet on all channels, if the transmission fails for a certain amount of time. If the packet is received by B on a channel different from A's current channel, A can defer transmission to B until a JOIN message is received.

A switching node needs to decide the duration of time it stays on a channel before switching to another channel. A node may potentially switch channels after sending each packet, but this results in performance loss due to the overhead of channel switching delay and LEAVE/JOIN message overhead. On the other hand, if a node stays on one channel too long, the delays for the packets on the other channel will be too high. So the duration for staying in one channel must be determined intelligently. This duration can be determined according to an estimation of traffic load in each channel. Designing adaptive channel scheduling algorithm is outside the scope of this dissertation. In the simulations, we use fixed duration of 50ms regardless of traffic load.

6.3.5 Force Mechanism

With the basic scheme of MCRP described above, it is possible that the destination node receives one or more RREQs, but all potential routes are determined to be infeasible. If the destination drops all these routes, it will result in the source failing to find a route although there is a path between the source and the destination.

In this case, the destination may still choose to select one route and send reply back, with the "force" flag set in the RREP packet. This option is called *force mechanism* and it is an optional feature of our protocol. The force mechanism guarantees that a route can be found if there is a path between source and destination.

If an intermediate node receives a RREP with the "force" flag on, then it is forced to choose the channel specified in the RREP, even if it is locked on another channel. The node selects channels according to the following rules. Suppose the RREP specifies channel x.

- Free node: The node becomes a locked node, and it is locked on channel x.
- Locked or Hard-locked node: If it is already locked on channel x, do nothing. If it is locked on another channel, send Route Error (RERR) packet on the forward and reverse path of the flows that were on the other channel, and then switch to channel x. The node state remains unchanged.
- Switching node: If one of its operating channels is channel x, then do nothing. If not, then choose one of its operating channel and send RERR on the forward and reverse path of the flow on that channel, and replace the channel with channel x in the set of its operating channels. The node state remains unchanged.

When a node receives an RERR message, it removes the corresponding route entry from the route table. The node state might change because of change in the route table, and the change is made according to the rules specified in Section 6.3.2.

The rules do not allow a locked node to become a switching node for a "forced" route, because it may cause two switching nodes in a flow.

There is at least one flow that loses the route because of this forced route. The route is considered to be broken, and the RERR is forwarded to the nodes on the route, as noted above. Then to recover from route error, the source node has to perform route discovery again. It is possible that two flows might force each other to break in the process of finding a route. To avoid the oscillation, nodes that have a route created by a force mechanism do not accept another RREP with the force flag on for a certain duration of time after it is created (we use 1 second in the simulations).

With the force mechanism, we can guarantee that a route can be eventually found if there is a path between source and destination. In the worst case, all routes will be converged into one common channel.

6.3.6 Route Maintenance

As mentioned earlier, routes expire if they are not used for a certain duration of time. When a node obtains a route, it sets up a timer. When the timer expires, the route is deleted from the routing table. Whenever a route is used, it is "refreshed" meaning that the timer is reset. Even when packets are sent on the forward path (the path from source to destination), both the forward path and the reverse path are refreshed. Due to node failures and mobility, a route may break while in use. If the MAC layer tries to transmit a packet several times without success, it tells the routing protocol that the link is broken. Then, the node that observes a link failure sends a RERR message to the source, so that the source can initiate a new route discovery process.

6.4 Performance Evaluation

We have evaluated the performance of our protocol MCRP through simulations using ns-2 simulator [33]. The metric we measure is the aggregate throughput over all flows. For each set of simulations, we run MCRP with 2, 3, and 4 channels, and also AODV with a single channel for the purpose of comparison.

We vary several parameters to see how various factors affect the performance of MCRP. The varying parameters used are: number of flows, flow rate, and connection pattern. The connection pattern is the set of source-destination pairs, and the connection pattern may affect the performance of MCRP significantly because channel allocation is done per flow.

For the traffic, we use constant bit rate (CBR) traffic, over UDP. The packet size is 512 bytes. Channel bit rate is 11Mbps. The network area is a square of 1000m x 1000m, and the transmission range of each node is approximately 250m. IEEE 802.11 DCF is used as the MAC protocol without any change.

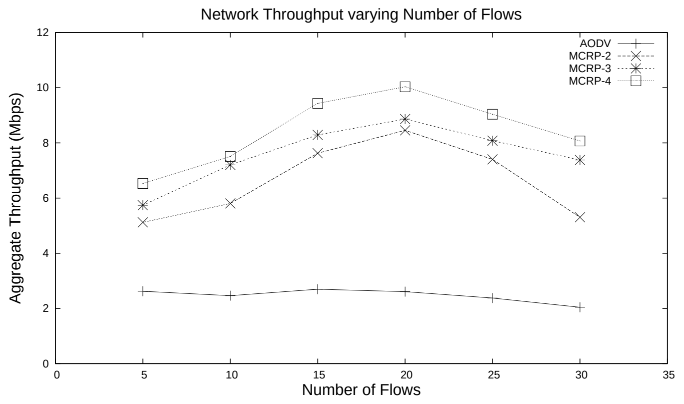


Figure 6.9: Network Throughput varying Number of Flows

In the first simulation, we measured the aggregate throughput varying the number of flows. 100 nodes are randomly placed in the area. Each flow generates traffic at 4Mbps. Since the flow rate is 4Mbps, the channel bandwidth is quickly saturated, as the number of flows is increased. The result is shown in Figure 6.9. The figure shows that MCRP can sometimes improve the throughput by a factor of 4 with 4 channels. Because of the routing protocol overhead and flow-level channel allocation, the performance of MCRP cannot reach the factor k improvement over a single channel protocol, where k is the number of channels. However, since the multiple channels distribute contention over the channels, the collision rate is reduced. That is why sometimes the throughput of MCRP with k channels is more than k times the throughput of a single channel protocol.

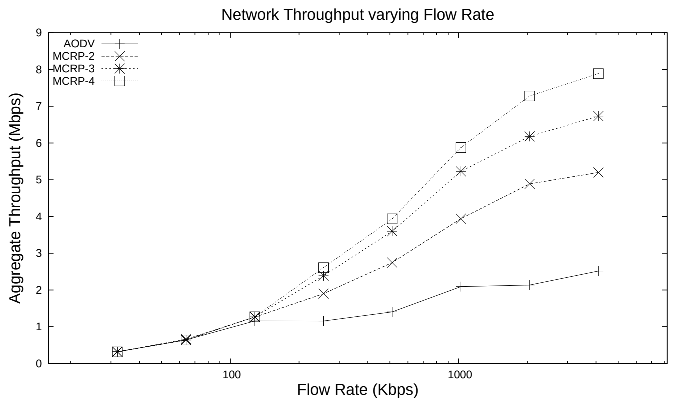


Figure 6.10: Network Throughput varying Flow Rate

In the next simulation, we varied the rate of each flow from 32 Kbps to 4096 Kbps. There are 50 nodes randomly placed in the area, and 10 flows having the same traffic generation rate. The results are shown in Figure 6.10. When the traffic load is low, having multiple channels does not make a difference because the total traffic is less than the bandwidth of a single channel. With low load, AODV can do a slightly better because of the overhead in MCRP, but as the graph shows, there is no significant difference. As the flow rate increases, the throughput of AODV increases slowly towards the limit of a single channel. As the traffic load becomes large, the performance improvement of MCRP over AODV becomes more significant.

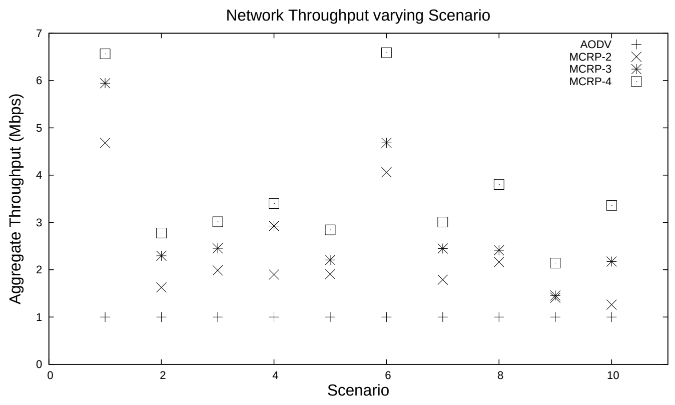


Figure 6.11: Normalized Aggregate Throughput varying Scenario

Finally, we simulated the protocol in different scenarios to see how the network topology and the location of source and destination affect protocol performance. In each scenario, 50 nodes are randomly placed in 1000m $\times$ 1000m area, and there are 10 flows with the traffic generation rate of 4Mbps. The result is shown in Figure 6.11, where aggregate throughput of protocols are normalized to the throughput of AODV (a single channel case). In most of the scenarios, MCRP with k channels achieves the throughput of a little less than k times the throughput of AODV, a single channel protocol. But sometimes the performance of MCRP is significantly higher than AODV (scenarios 1 and 6), even more than factor of k. This is because MCRP reduces collisions by distributing contention among channels.

The simulation results show that MCRP significantly improves the network throughput by utilizing multiple channels, even with a single transceiver at each node.

6.5 Summary

In this chapter, we considered a network layer approach for utilizing multiple channels in wireless ad hoc networks, assuming a single-channel MAC protocol and single transceiver at each node. We identified that assigning channels to nodes can result in significant performance degradation due to the deafness problem. On the other hand, assigning nodes to flows without allowing dynamic channel switching can result in low utilization of channels and also reduce the route discovery success ratio. Finally, to avoid deafness problem, two consecutive nodes in a path should not be both switching nodes.

Based on the discussions, we proposed a MCRP, a routing protocol that utilizes multiple channels to improve network throughput. MCRP works with a single transceiver, and a single-channel MAC protocol such as IEEE 802.11 DCF. Since MCRP does not require additional hardware or a new MAC protocol, it can be easily deployed on currently used devices. The simulation results show that MCRP improves network throughput substantially by efficiently allocating channels to flows.

Chapter 7

A Link-layer Channel Assignment and Scheduling Protocol for Multi-Channel Networks

In Chapter 3, we have proposed MMAC, a multi-channel MAC protocol for multi-hop networks. Also, in Chapter 6, we proposed a multi-channel routing protocol called MCRP. As discussed in Chapter 5, MMAC tends to assign a single channel to all nodes participating in a flow, and also nodes in the overlapping flows. This behavior significantly limits the channel utilization, and we proposed an extension to AODV that tries to find disjoint flows. In MCRP, overlapping flows can be assigned different channels, by making the overlapping node a switching node. Within the constraint that two consecutive nodes cannot become switching nodes, MCRP has more freedom than MMAC in assigning channels to flows.

However, both protocols do not utilize multiple channels within a flow. The benefit of assigning a common channel for all nodes within a flow is that nodes do not need to switch channels. However, this does not mean that nodes never switch channels, because in protocols such as MCRP, some nodes switch channels to serve multiple flows. There are several disadvantages of assigning a common channels to a flow. First, The flow throughput suffers from "self-contention", a contention within a flow [56]. For example, consider a chain topology with four nodes, A, B, C, and D. If A is sending packets to D via B and C, nodes A, B and C contend for channel, which degrades the flow throughput. Second, for flows that span over large number of hops, the level of channel contention may be different in different regions. For example, channel 1 may be the best channel on one end, while channel 2 is the best channel on the other end. If a common channel is used for the entire route, the best channel in the current region cannot be utilized.

In this chapter, we develop a link-layer channel assignment protocol that can allocate different channels within a flow. The fundamental difficulty of channel assignment and coordination comes from the fact that nodes are equipped with a single interface. If nodes are equipped with two interfaces, the channel assignment becomes much simpler [16], and it becomes easy to avoid assigning a common channel to a flow. Thus, we emulate multiple interface with a single interface by switching channels periodically. If interfaces switch channels in a synchronized manner, the channel assignment algorithm can assign channels as if each node is equipped with two interfaces (and these two interfaces do not switch channels). The proposed protocol called LCAS is a link-layer protocol, that sits between routing and MAC protocols. LCAS requires clock synchronization.

In the rest of the chapter, we first discuss challenges in multi-channel networking, and present basic insights in designing our proposed protocol, LCAS. After that, we describe the details of LCAS. Then, we report results from performance evaulation of the proposed protocol using ns-2 simulations. Finally, the chapter is concluded with a summary.

7.1 Preliminaries

In this section, we develop the basic insights behind the design of LCAS. Note that the environment we consider is a multi-channel multi-hop network, where nodes are equipped with a single interface. Before discussing LCAS, we first briefly review two other protocols designed for single-interface environment, MMAC and SSCH [57], to gain insights for designing the LCAS protocol.

7.1.1 MMAC

MMAC is described in detail in Chapter 3. The channel assignment method in MMAC is to assign channels to links, because sender and receiver collaboratively decide upon a channel. The channel scheduling method in MMAC is to assign channels to each node for a fixed duration of time (beacon interval), and reassign channels in every beacon interval according to current traffic demand. Since there is no pattern in switching channels, the next schedule (the selected channel for the next beacon interval) has to be notified to the neighbors in every beacon interval. Since the next schedule is decided based on current traffic demand at each node, a node always schedules all links that have traffic on the next slot. Because of this behavior, multi-hop flows often cannot benefit from channel diversity. This behavior is described using the following example, illustrated in Figure 7.1.

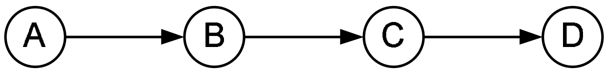


Figure 7.1: A network scenario with 4 nodes. There is a traffic flow from node A to node D.

In this figure, node A is sending packets to node D via node B and C. The ideal scheduling in this scenario is as follows. Suppose time is time is divided into equal-sized intervals. At ith interval, node A sends packets to B on channel 1, and C sends packets to D on channel 2. At $(i+1)$th beacon interval, node B sends packets to node C on channel 3. With this scheduling, the throughput of the flow is $W/2$, where W is the channel capacity. (If all links are scheduled on the same channel, the throughput is $W/3$, since only one link among three links can be active at a time.)

In MMAC, since all links have traffic demand, nodes try to schedule all links in every interval. At ith beacon interval, suppose node A sends the ATIM message to node B first. Node B selects channel 1. After that, node B sends an ATIM message to node C. Since node B has already selected channel 1, node C has to select channel 1, according to the protocol. After that node C sends an ATIM message and node D selects channel 1. So in this interval, all links are assigned the same channel. At $(i + 1)$th beacon interval, suppose node A and node C sends ATIM messages before node B. Then, link AB can be assigned on channel 1, and link CD can be assigned on channel 2. When node B sends an ATIM message, node C rejects the message and link BC cannot be assigned a channel for the given interval. Since the order of sending ATIM messages is random in each interval, multi-hop flows often cannot benefit from channel diversity. This problem is reflected in the simulation results, shown in Section 7.3.

7.1.2 SSCH

In SSCH [57], time is divided into slots, and each node hops from channel to channel in every slot according to a pseudo-random sequence. The pseudo-random sequence is generated using a starting channel and a random seed. A node can have multiple (channel, seed) pairs, and these pairs are used in a round robin manner to compute the channel for the next slot. For example, suppose node A switches channels according to two (channel, seed) pairs, (1, 2) and (4, 3). In the first slot, node

A uses channel 1, according to the first (channel, seed) pair. In the next slot, node A uses channel 4, based on the second (channel, seed) pair. In the third slot, node A uses a channel based on the first (channel, seed) pair, but this time the selected channel is different than channel 1, because it is changed using the seed. In SSCH, nodes simply add the seed value to the channel, so in the third slot, the selected channel is channel 3. If there are 10 channels, after selecting channels 1, 3, 5, 7, and 9, the node returns to channel 1. In [57], the authors use 4 (channel, seed) pairs.

The pseudo-random sequence is designed so that every pair of neighboring nodes meet at a common channel after every finite number of slots. During that slot, the nodes can exchange messages. The (channel, seed) pairs are known to neighbors by exchanging messages, so a node knows when it will meet with its neighbor.

When node A has to continuously send packets to node B, node A cannot rely on only the slots where A and B meet. In this case, node A modifies one or multiple of its (channel, seed) pairs to match that of node B, so that the number of overlapping slots increases.

Let us consider the scenario in Figure 7.1. Suppose each node hops according to 4 (channel, seed) pairs (as used in [57]). Since node A has packets for B, node A modifies two of its 4 (channel, seed) pairs to match that of B. After that, node B marks the slots where it is receiving packets from A as "receiving" slots, and modifies the other two (channel, seed) pairs to match that of C. Finally, node C marks the slots where it is receiving packets from B as "receiving" slots, and modifies the other two (channel, seed) pairs to match that of D. With this scheme, the protocol achieves a throughput of $W/2$, which is ideal.

The channel assignment in SSCH is to assign channels to nodes, because each node chooses its own (channel, seed) pairs, which correspond to hopping sequences. The problem with this approach is that the sender can follow the receiver's sequence, but there is no collaborative process between sender and receiver in deciding a sequence. Consider the scenario in Figure 7.2. There are 5 nodes in the network, and 4 single-hop flows. Let us assume there are more than 4 channels available, so that every link can be potentially assigned a different channel. The ideal channel scheduling in this scenario would be to have link AB and link CD on the same slot, and AC and DE on another slot (this strategy achieves throughput of $2W$). However, in SSCH, node A and node D choose which slots they should synchronize with their receivers independently. Since both A and D have traffic for node C, it is quite possible that when A and D modify their (channel, seed) pairs to follow C's channel sequence, they might end up following C's channel sequence in the same time slots. In this case, the aggregate throughput is $3W/2$, whereas the ideal throughput is $2W$.

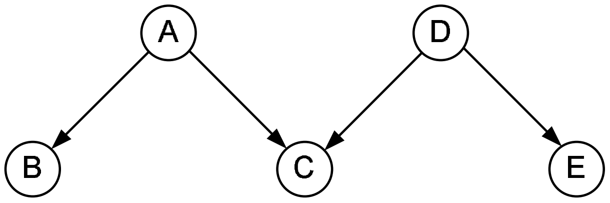


Figure 7.2: A network scenario with 5 nodes. Each arrow indicates a traffic flow.

Because of the channel assignment scheme, SSCH may not efficiently assign channels in scenarios where there are multiple flows, and these flows are overlapping at nodes.

7.1.3 Our Approach

From previous discussions, we argue that to achieve best performance, the protocol should be able to assign time slots to links, so that it can decide which links should be active at which time slots, as done in SSCH (by having a pair of nodes selecting a common channel for certain time slots). Also, the channel assignment scheme should assign channels to links, by having the two end-nodes collaborate with each other as in MMAC. Here we present an overview of LCAS. The detailed description of the protocol is in Section 7.2.

In LCAS, we divide time into intervals, and each interval into two time slots, slot 1 and slot 2. (The reason for dividing an interval into two slots will be discussed later.) All slots are of the equal size. The size of a slot is a protocol parameter, but to reduce the frequency of channel switching, a node should be able to send multiple packets in a slot. In our simulations, we use 50ms for the slot size.

A node is assigned a channel in each time slot. If node A chose channel 1 for slot 1 and channel 2 for slot 2, then node A follows the same schedule in each interval, as in Figure 7.3.

LCAS can be divided into two parts: channel scheduling and channel assignment. First, the protocol assigns time slots to links, and then chooses which channel will be used on each link. Suppose node A and node B choose to communicate in time slot 1 using channel 1. Then, node A and node B are assigned channel 1 for slot 1. If other links adjacent to A (links that have A

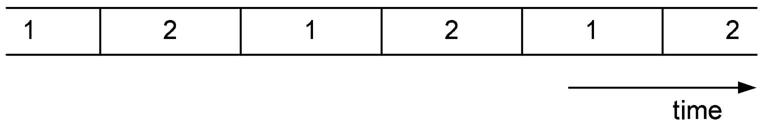


Figure 7.3: An example channel schedule. The node switches between channel 1 and channel 2, in a round-robin manner.

on its one end) or B are assigned slot 1, these links must use channel 1. If we have two slots in each interval, only two channels (one for each slot) can be assigned to all links adjacent to a node. The channel scheduling and channel assignment is both done in a distributed manner, as will be described in Section 7.2.

Consider the scenario in Figure 7.2 above. If node A is currently sending packets to B or receiving packets from B, then both A and B consider the link AB as an *active link*. Otherwise, the link is considered as an *idle link*. All idle links are assigned a common channel, say channel 1, for all slots. On the other hand, each active link is assigned a time slot, either slot 1 and slot 2. Suppose all 4 links in this scenario are active links. Then the goal of channel scheduling protocol is to assign the slots as illustrated in Figure 7.4, because the scheduling can lead to the maximum throughput. If there are large number of channels so that links AB and CD can be assigned on different channels and links AC and DE can also be assigned on different channels, the aggregate throughput will be $2W$, which is the maximum throughput achievable in this scenario. The channel scheduling algorithm in LCAS tries to achieve the best channel schedule, in a distributed way.

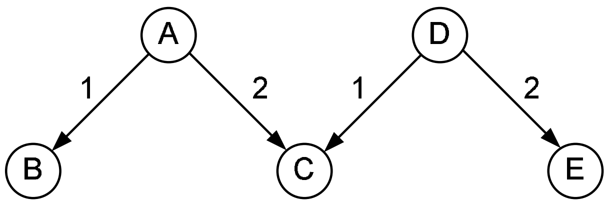


Figure 7.4: A channel scheduling example for the scenario in Figure 7.2. This schedule achieves the maximum throughput.

As mentioned earlier, LCAS divides time into equal-sized slots and assigns channels on these slots. Because slot length is fixed, bandwidth may be wasted if traffic load on the links are unbalanced. For example, in Figure 7.4, time is equally shared between link DC and link DE due to equal sized time slots. If the load on link DC is $0.8W$ (where W is the channel bit-rate) and the load on link DE is $0.2W$, bandwidth of $0.3W$ will be unnecesarily wasted while node D is on channel 2. To deal with unbalanced load, the slot time must be adapted to the current traffic condition. LCAS does not address this issue.

There may be cases where a node has more than two adjacent links that are active. Consider the scenario in Figure 7.5. In this scenario, each arrow indicates a separate single-hop flow that is currently active. Since nodes C and D have 3 adjacent active links, they have to assign two links on the same slot. If two adjacent links are assigned on the same slot, they have to be assigned on the same channel, otherwise the channel mismatch problem described in Section 3.2 may arise. This is the constraint that the channel scheduling scheme puts on the channel assignment algorithm.

Under this scenario, there are several schedules that produce the maximum throughput, and one example is illustrated in Figure 7.5. The numbers on the links are assigned time slots. Since links AB and BC are assigned on the same slot, they have to be assigned on the same channel, and it is the same case for links DE and DF. If links AB and BC are assigned channel 1, link CD is assigned channel 2 and link DE and DF are assigned on channel 3, then the aggregate throughput here is $3W/2$.

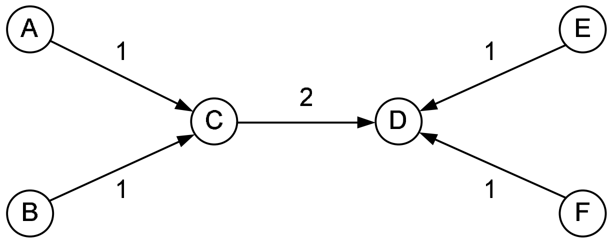
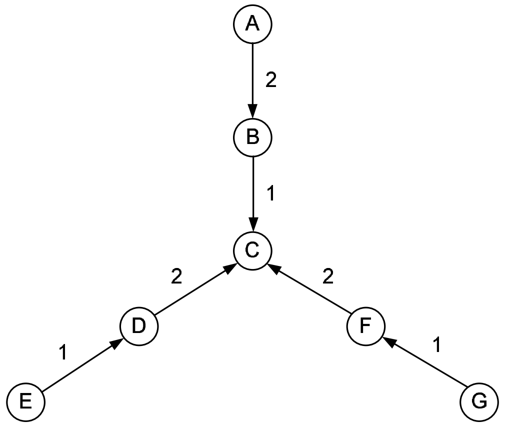


Figure 7.5: A network scenario where nodes have more than 2 adjacent links.

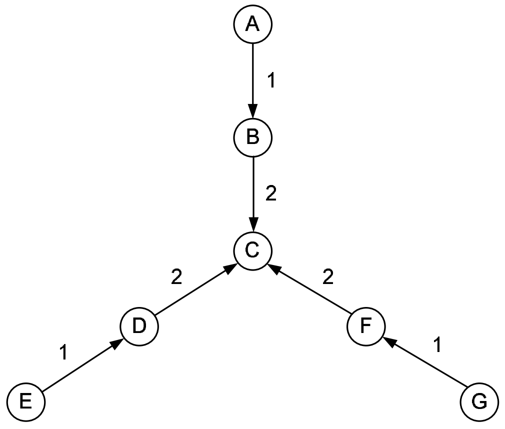
Note that if there is only a single active link between two nodes, both slots are assigned to the link, so that the channel bandwidth is not wasted. In Figure 7.5, if link CD is the only active link, the link is assigned both slots.

In general, the goal of our channel scheduling protocol is to assign time slots to active links, so that when we divide the network graph into two subgraphs according to the set of edges in each slot, *the number of disjoint component is maximized.* The motivation behind the goal is as follows. If there are large number of channels so that every disjoint component can be assigned a different channel, then the number of disjoint components translates into number of simultaneous transmissions in each time slot, because transmissions from different components do not interfere with each other. (It is possible that if a component is large, spatial reuse can allow simultaneous transmissions to take place within a component.) Even if there are limited number of channels, generating larger number of disjoint components gives more freedom to selecting channels based on channel quality, because all the links in a component must choose the same channel.

Consider the scenario in Figure 7.6. There are six links that are active. The maximum number of disjoint components in this network is 5, because node C must assign two of its adjacent links on the same slot. The schedule in Figure 7.6(a) generates 5 components, whereas the schedule in Figure 7.6(b) generates 4 components. If there are 3 channels available, the schedule in Figure 7.6(a) can achieve a throughput of $5W/2$ (by assigning links AB and BC on channel 1, links CD, CF on channel 2, link DE on channel 3 and link FG on channel 4). On the other hand, the schedule in Figure 7.6(b) achieves a throughput of $2W$.



(a) A schedule that achieves $5W/2$



(b) A schedule that achieves $2W$

Figure 7.6: A network scenario to illustrate channel scheduling. The goal of channel scheduling is to maximize the number of disjoint components. The number on the links are slot numbers.

The channel scheduling problem can be mapped to an edge coloring problem with insufficient colors. Since we divide an interval into two slots, assigning time slots to links is same as coloring the edges of a graph with two colors. In general, with just two colors, the graph cannot be colored so that no two adjacent edges are colored with the same color. The purpose of edge coloring here is not to have every adjacent edge colored with a different color, but to produce the maximum number of disjoint components, when the graph is divided into subgraphs according to edge colors. It is intuitive that when more colors are used, the number of disjoint components can increase. If there are enough colors, the number of disjoint component will equal the number of edges in the graph.

Let us assume that there are large number of channels, so that each component can be assigned a different channels. Also, assume that there is no spatial reuse within a component, so that only one transmission can take place at a time. With these assumptions, we can compute the aggregate throughput as follows. Suppose there is a connected graph $G(V,E)$ (if the network topology is disconnected, we can deal with each connected graph separately). We use C colors to color the edges. After coloring the edges, we divide the graph into C subgraphs, according to edge colors. Let us denote a subgraph with edges colored with color i as ${G_C}^i$. Also, we denote the number of disjoint components in ${G_C}^i$ as $D({G_C}^i)$. Then, the aggregate throughput of the graph, $T(G_C)$ will be

$$T(G_C) = \frac{\sum_i D({G_C}^i)}{C} W, \qquad i = 1 \ldots C \tag{7.1}$$

where W is the channel bandwidth.

Now we prove that when edge-coloring a connected graph with any number of colors C, the maximum throughput ratio it can achieve over edge-coloring with two colors is 2.

Lemma 1: For any connected network, scheduling edges with C slots cannot achieve more than twice the aggregate throughput compared to scheduling edges with two colors.

Proof: Suppose we edge-color a connected graph with C colors, and divide the network into C subgraphs according to the edge colors. For any color i, the maximum number of disjoint components is $\lfloor n/2 \rfloor$, because a disjoint component needs at least two vertices that are not participating in other components. So for any connected graph and any number of colors,

$$\frac{\sum_i D({G_C}^i)}{C} \leq \frac{\lfloor n/2 \rfloor \times C}{C} = \lfloor n/2 \rfloor \tag{7.2}$$

If C-coloring can achieve $\lfloor n/2 \rfloor$ components, that means we can find at least one set of $\lfloor n/2 \rfloor$ edges in which no vertex appears twice. So with 2 colors, one can color these edges with a color, and color all other edges with another color. Then,

$$\frac{\sum_i D({G_2}^i)}{2} \geq \frac{\lfloor n/2 \rfloor}{2} \tag{7.3}$$

So the throughput ratio between two coloring scheme is,

$$\frac{T(G_C)}{T(G_2)} \leq 2 \tag{7.4}$$

Suppose a set of $\lfloor n/2 \rfloor$ disjoint edges cannot be found in the graph, and r is the maximum number of disjoint edges that can be found in the graph. In this case, applying the same coloring scheme,

$$\frac{\sum_i D({G_2}^i)}{2} \geq \frac{r}{2} \tag{7.5}$$

If we color this graph with C colors, no subgraph can have more than r disjoint components. So,

$$\frac{\sum_i D({G_C}^i)}{C} \leq \frac{r \times C}{C} = r \tag{7.6}$$

From these two equations we get a bound on the throughput ratio as in 7.4. Thus, scheduling a connected graph with C slots can achieve no more than twice the aggregate throughput compared with scheduling the graph with 2 slots.

This result indicates that increasing number of slots does not increase the aggregate throughput linearly. Whereas, channel switching overhead and control message overhead increases linearly with the number of slots. Thus, we choose to use two slots. Note that the protocol can always be extended to use more slots.

When two end-nodes choose a slot for a link, they choose a channel based on the neighbor information. To assign a channel to a link, the two end-nodes exchange messages to agree on a channel, which is used by minimum number of nodes in the range of interference. When counting the number of interfering links, only the links assigned on the same slot are counted. As mentioned above, there is an important constraint in assigning channels, that is, consecutive links on the same slot need to be assigned the same channel. (If a new link finds that its consecutive link is assigned the same slot, it has to select the same channel as the consecutive link.)

7.2 LCAS: The proposed protocol [4]

In Section 7.1, we discussed the basic design of the proposed protocol, LCAS (Link-layer Channel Assignment and Scheduling). In this section, we describe the details of how the basic design is implemented into a protocol.

7.2.1 Neighbor Discovery / Hello Messages

In LCAS, a node selects its channels according to information collected from neighbors. Thus, it has to know all of its neighbors. Also, whenever the channel assignment of a node changes, the information has to be updated at its neighbors, so that they can make decisions with up-to-date information.

There are two approaches for exchanging neighbor information: proactive and reactive. In a proactive approach, nodes broadcast *hello* messages to their neighbors periodically. Whenever a node broadcasts a message, the node has to broadcast it on all channels. (Broadcasting will be discussed further later.) The hello message includes the node id and the selected channels. For example, if node A has selected channel 1 for slot 1 and channel 2 for slot 2, the hello message will have (A, 1, 2).

When a new node joins the network, it can collect its neighbor information by waiting for a single hello period. Whenever a node changes its channels, it has two options. It can immediately send a hello message and start a new hello period, or it can wait for the next hello period to send the update message. The two schemes have trade-off between speed of update and overhead.

Another approach for exchanging neighbor information is a reactive approach. In a reactive approach, a node broadcasts a message only when the node changes its assigned channels. When a new node joins in, it broadcasts a *JOIN* message on the network. Whenever a node receives this JOIN message, it replies back by sending an *WELCOME* message back to the newly joined node.

When a new node joins, by default it chooses channel 1 for both slots. When broadcasting a JOIN message, it broadcasts the message on all channels, one by one. The nodes that receive the

JOIN message send back a WELCOME message on channel 1, so that the new node can receive the packet. There can be multiple nodes sending WELCOME messages, and the collision avoidance is taken care by underlying MAC protocol. Note that our link-layer protocol runs on top of a MAC protocol, such as IEEE 802.11 DCF.

7.2.2 Assigning Slots to Links

When a node joins the network, it selects a common channel (channel 1) for both of its slots. Using hello messages or reactive neighbor discovery process (using JOIN and WELCOME messages), it collects the neighbor information, and stores the information in a table. (In the simulations, we use both hello messages and neighbor discovery process to collect neighbor information.) Suppose node E joins the network and collects neighbor information in Figure 7.7, and creates a *neighbor table* as shown in Figure 7.8.

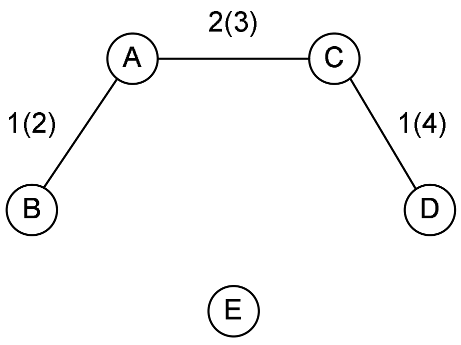


Figure 7.7: A network scenario with 5 nodes. The numbers on the edges indicate the assigned slot (channel). When node E joins, it collects the neighbor information and creates a neighbor table as shown in Figure 7.8.

In Figure 7.7, the numbers on a link indicate the assigned slot (channel). For example, link AC is assigned on slot 2, and channel 3. Thus, nodes A and C have to stay on channel 3 for slot 2. Note that a node may have an idle slot. In this scenario, node B does not have any link where it can send or receive traffic in slot 2. In this case, B sets the "IDLE" flag for the slot in the HELLO packet so that its neighbors know that the slot is idle.

Suppose node E starts a traffic flow for node D. Then, a new active link should be created between D and E. To assign a slot and a channel for the link DE, node E sends a *CONNECT* message to node D, on D's channel. So if E sends the packet in slot 1, E sends the packet on

Neighbor Table of E

ID	Slot 1	Slot 2
A	2	3
B	2	1 (Idle)
C	4	3
D	4	1 (Idle)

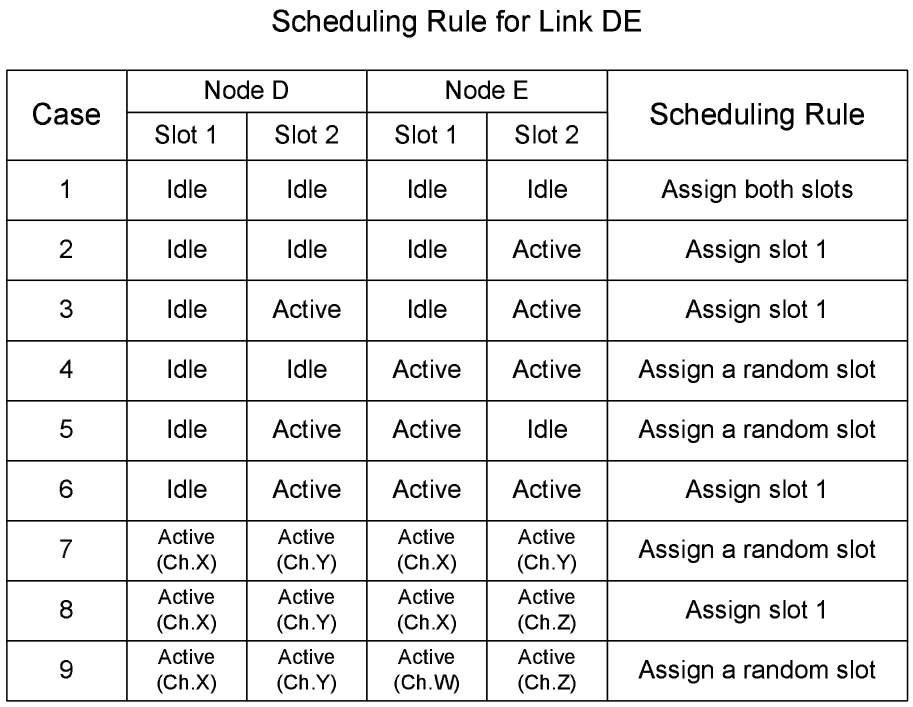
Figure 7.8: The neighbor table of node E in Figure 7.7. It indicates what channels the neighbors will use in each slot. When the slot is marked "Idle", it means the node does not have any traffic flow in that particular slot, so it can freely switch the channel for the slot.

channel 4 (switching channels within a slot is discussed further later). When node D receives the CONNECT packet, it selects a slot and channel for link DE and sends the information back to E in a *CONNECT-ACK* packet.

When node D receives a CONNECT packet from node E, it first selects a time slot for link DE. It selects the a slot for the link according to the rules in Figure 7.9. Note that the goal of channel scheduling is to produce as many disjoint components as possible in a connected graph. So the rule is to select a slot which is expected to result in a larger number of disjoint components than the other slot.

Note that in case 9, channels between D and E do not match for either slots. In this case, node E (that sends the CONNECT message) has to switch its channel. Suppose E chooses slot 1 for link DE. Then, node E has to use channel W instead of channel X. When link DE is assigned channel W, adjacent links that were using channel X must also switch to channel W. Before switching channels, node E broadcasts a SWITCH message on channel X. When a node receives the SWITCH message on channel X, it forwards the SWITCH message and then switches its channel to channel W.

For link DE in the above example (in Figure 7.7), D has slot 1 active and slot 2 idle, and both of E's slots are idle. So according to the rules, link DE is assigned on slot 2.


Scheduling Rule for Link DE

Case	Node D		Node E		Scheduling Rule
	Slot 1	Slot 2	Slot 1	Slot 2	
1	Idle	Idle	Idle	Idle	Assign both slots
2	Idle	Idle	Idle	Active	Assign slot 1
3	Idle	Active	Idle	Active	Assign slot 1
4	Idle	Idle	Active	Active	Assign a random slot
5	Idle	Active	Active	Idle	Assign a random slot
6	Idle	Active	Active	Active	Assign slot 1
7	Active (Ch.X)	Active (Ch.Y)	Active (Ch.X)	Active (Ch.Y)	Assign a random slot
8	Active (Ch.X)	Active (Ch.Y)	Active (Ch.X)	Active (Ch.Z)	Assign slot 1
9	Active (Ch.X)	Active (Ch.Y)	Active (Ch.W)	Active (Ch.Z)	Assign a random slot

Figure 7.9: Rules for scheduling slots to links. The new link is between node D and E, and the scheduling rule applies to the new link based on status of the two end nodes.

7.2.3 Assigning Channels to Links

Once a link is assigned a slot, a channel should be assigned to the link. The node which receives the CONNECT packet selects a channel for the link, using the channel information of the sender and the receiver.

The channel assignment for a link is affected by the link's schedule. In Figure 7.9, links that fall in cases 1 through 3 can freely choose a channel. For the other cases, the channel is already chosen because there is an adjacent link that is assigned on the same slot. If a link can be assigned any channel, a channel that is used by minimum number of neighbors on the same slot can be chosen.

If the new link falls in case 9, the link cannot be assigned a channel, because both slots are active on both end nodes, and they are assigned different channels. This scenario is illustrated in Figure 7.10.

In this scenario, suppose node C has traffic for node D. Since both slots at node C and node D are active and the links are assigned on different channels, the link CD cannot be scheduled a channel in either slot. In this case, when node C sends a CONNECT message to node D, D selects

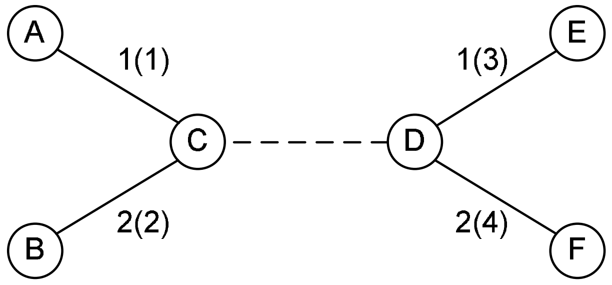


Figure 7.10: A network scenario where a link cannot be assigned a channel without changing channels of other links. Numbers on each link indicate slot(channel) assigned for the link.

a random slot, and the channel that has been already assigned to the slot. For example, in Figure 7.10, if node D chooses slot 1 for link CD, it selects channel 3 for the link. Then, link AC has to be changed to channel 3. To change channels, node C sends a SWITCH message to node A saying it should switch its channel for slot 1 to channel 3. When A receives this message, A switches its channel to channel 3 from the next slot 1. If A has other adjacent links that are scheduled on slot 1, it propagates the SWITCH message, so that eventually all links on the component is switched to channel 3. The result of inserting link CD is illustrated in Figure 7.11. Note that as discussed earlier, whenever a node switches its channels, it broadcasts the updated information in a HELLO message.

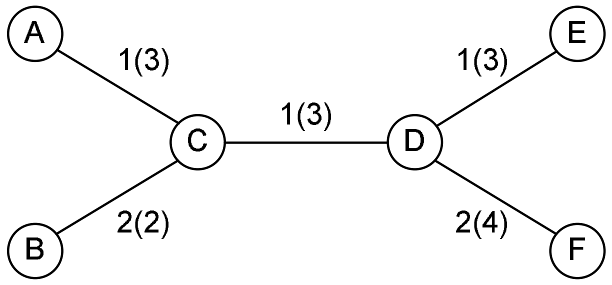


Figure 7.11: The channel schedule and assignment after link CD has been inserted in Figure 7.10.

It is always possible that HELLO messages are lost. In this case, node A might think node C will be on channel 1 during slot 1, when C has switched to channel 3. Then, node A might try to find node C using the neighbor discovery process, or it can wait for the next HELLO message from C if HELLO messages are used.

7.2.4 Broadcasting / Switching Channels within a Slot

Even though nodes mostly follow their channel schedules, there are situations where a node inevitably needs to switch channels within a slot. For example, when a node needs to broadcast a packet, it has to switch channels so that it can send the packet on all channels. Also, when a node wants to start a flow with a neighbor that it does not share a common channel in either slot, the node has to switch channels to communicate with the neighbor so that it can establish an active link between the two nodes.

When a node switches channels in a slot, the channel mismatch problem discussed in Section 3.2 can arise. To avoid this problem, we apply the following rules. First, whenever a node switches to another channel in a slot, it first broadcasts a "LEAVE" message on its original channel, before switching. After the node finishes communication on another channel and returns to its original channel, it broadcasts a "RETURN" message on its original channel so that its neighbors on the channel know that the node has returned. This rule prevents the channel mismatch problem for the neighbors that are sharing a common channel with the node in the slot. However, nodes that are on different channels cannot receive LEAVE or RETURN message. Also, LEAVE and RETURN messages can be lost. Let us consider the scenario in Figure 7.12. Suppose on slot 1, node A is assigned channel 1, B is assigned channel 2, and C is assigned channel 3. When B switches to channel 3 to send packets to C, A might switch to channel 2 in order to communicate with B. In this case, A would not have heard the LEAVE message. To address this situation, additional rules are necessary.

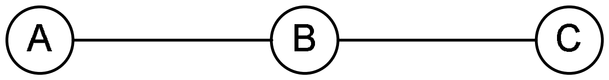


Figure 7.12: A network scenario with 3 nodes.

- If a node receives packets from a neighbor who has switched out of its original channel, the node does not switch channels for the current slot. In the above scenario, if node B receives packets from A, B does not switch channels for that particular slot. The motivation for this rule is to allow the switched node to successfully send packets.

- If a node successfully communicates on a channel different than its original channel, it does not switch channels in the same slot of the next interval. However, if a node fails to communicate on a different channel, it can retry in the next interval. In the above scenario, if B successfully sends packets to C on channel 3, it refrains from switching channels in the next interval, so that A can have a chance to send packets to B on channel 2. The motivation behind this rule is to avoid starvation. If B keeps on switching to channel 3 in every slot, A cannot send packets to B on channel 2. With this rule, if node A does not succeed in sending packets to B on channel 2, it will be able to send packets for B on channel 2 in the next interval.

7.3 Performance Evaluation

We have performed simulations to study the performance of our proposed protocol, LCAS. In this section, we report and discuss the results.

7.3.1 Simulation Setup

For simulations, we have used the ns-2 simulator [33]. The number of channels is varied in the simulations, but the bit rate of each channel is fixed to 11Mbps. The transmission range of each node is 250m. For traffic, FTP over TCP and CBR (Constant Bit Rate) over UDP are used. UDP traffic is not sensitive to packet delay, whereas TCP traffic is affected by packet delay and losses. Since the channel scheduling method of LCAS creates variation in packet delay, it is important to study the performance under TCP traffic. In addition to number of channels, number of nodes and flows are also varied as parameters. We run simulations on a grid topology and a random topology. The grid topology is useful for examining specific behaviors of LCAS, whereas random topology simulation is necessary to study the feasibility of LCAS in arbitrary situations.

Under these parameters, we simulated LCAS on top of IEEE 802.11 DCF. For comparison, we also simulate IEEE 802.11 DCF which uses a single channel, and MMAC. Since MMAC is also a multi-channel link-layer protocol for multi-hop networks, we can find out how LCAS performs well compared to a protocol developed for the same environment.

Other than MMAC, SSCH [57] is another link-layer multi-channel protocol designed for single-interface nodes. Here we compare SSCH and LCAS qualitatively. As discussed in Section 7.1, the major difference between SSCH and LCAS is in the channel assignment strategy. SSCH assigns channels (hopping sequences) to nodes, whereas LCAS assigns channels to links. So in SSCH, a sender follows the hopping sequence of a receiver whenever it needs to communicate with the receiver. As shown earlier in the example in Figure 7.2, this strategy may result in inefficient channel assignment, if multiple flows are overlapping at nodes. LCAS assigns channels to links by having the two end nodes collaboratively decide a channel. This strategy often results in a more efficient channel assignment. The two protocols have similarities in other aspects, such as dividing a time interval into multiple slots and scheduling the slots in a round-robin manner. Because of the channel assignment strategy, LCAS can achieve a better channel utilization in scenarios where many flows are overlapping with each other. For other scenarios, the performance of the two protocols can be comparable to each other.

For MMAC and LCAS, we take into account channel switching and synchronization overhead. To synchronize clocks, existing schemes such as [58] can be applied. In terms of synchronization, we make similar arguments as SSCH [57]. In [57], switching and synchronization overhead accounted for 3.6% of the total bandwidth. In our simulations, we assume that 5% of the bandwidth is used for switching and synchronization overhead, and reduce 5% from the channel bit rate.

7.3.2 Results

Grid Topology

For the first set of simulations, we ran the protocols in a grid topology. 64 nodes were placed on the 8 × 8 grid points, as in Figure 7.13. The distance between two neighboring nodes is 250m, which is the transmission range of a node.

Impact of network contention First, we measured the aggregate throughput varying number of flows. We fixed the number of hops in each flow to 3. For each flow, a source is randomly chosen among all nodes, and its destination is randomly chosen among the nodes that are 3 hops away from the source. (Later, when we simulate random topology, flows have random hop distance between source and destination.) The number of channels is set to 12, as in IEEE 802.11a. The number of flows is varied from 1 to 10, and each flow runs CBR traffic of 4Mbps. The result is

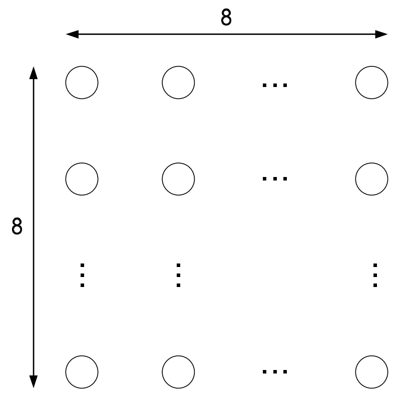


Figure 7.13: The 8 × 8 grid topology used in the simulations.

shown in Figure 7.14. As the number of flows increase, the throughput of LCAS increases faster than MMAC and DCF. When the number of flows becomes large, flows start to overlap with each other at some nodes. In this situation, LCAS still uses channel scheduling to distribute load among channel, but MMAC cannot efficiently do that, because MMAC tends to assign a common channel for all overlapping flows. As a result, LCAS achieves around 150% improvement over MMAC with 10 flows. Even when the number of flows is small, LCAS assigns different slots and channels to links within a flow to increase the flow throughput. When the number of flows is small, MMAC can achieve lower throughput than DCF due to channel negotiation overhead.

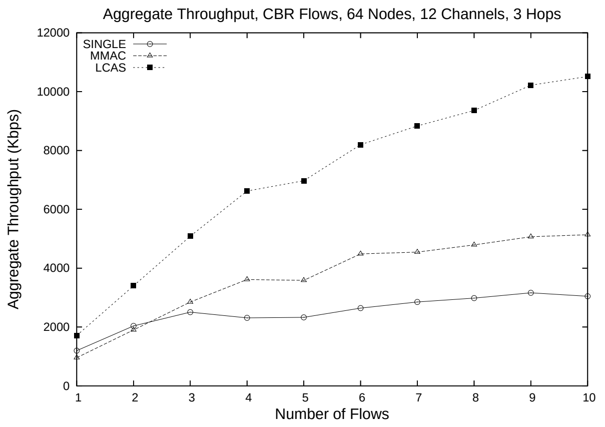


Figure 7.14: Aggregate throughput varying number of flows.

Impact of number of hops between flows Next, we varied the number of hops between source and destination in the flows. This experiment is to see how LCAS reduces contention within a flow by assigning different channels to different links. There are 12 channels available, and 8 flows run

CBR traffic at 4Mbps each. The result is shown in Figure 7.15. The throughput decreases as the number of hops between source and destination increases, because more transmissions need to be made in order to obtain the same throughput. As the number of hops increases, the performance of MMAC degrades quickly, and after a certain point, its performance is similar to DCF. However, LCAS achieves higher throughput even when the hop distance of flows is large. This is achieved by scheduling links within a flow and assigning different channels so that contention within a flow is reduced.

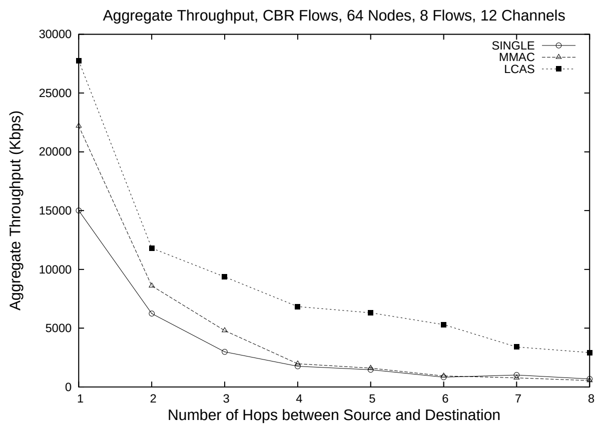


Figure 7.15: Aggregate throughput varying number of hops between source and destination.

Per-flow Throughput In this simulation, we have measured per-flow throughput to see if LCAS is starving any flows to achieve higher aggregate throughput. In the simulated scenario, we used 12 channels and generated 10 CBR flows, at 4Mbps each. The number of hops between source and destination is 3. Figure 7.16 shows the result. The result shows that LCAS does not starve a flow in order to achieve higher aggregate throughput, but increases per-flow throughput in each flow.

Impact of Number of Channels: Until now, the number of channels was fixed to 12. This experiment is to study the impact of number of available channels. In the simulated scenario, 8 flows were created so that the hop distance between source and destination is 3. Each flow runs a CBR flow at 4Mbps. Under these settings, we varied the number of channels from 1 to 8. The result is shown in Figure 7.17. The result shows that as the number of available channels increase, LCAS can utilize more channels to improve throughput, compared to MMAC.

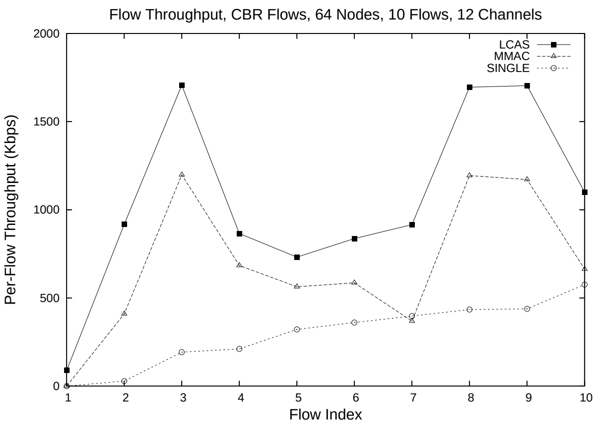


Figure 7.16: Per-flow throughput in a scenario with 10 flows.

As we will see in the random topology simulations, the benefit of LCAS over MMAC is more substantial when the number of channels is large. This is mainly because LCAS reduces contention within a flow and among overlapping flows through channel scheduling. When the number of available channels is small, the impact of these contentions diminishes, and the channel assignment approach in MMAC (assigning a common channel for all nodes in a flow and also overlapping flows) already achieves high channel utilization. On the other hand, when the number of channels is large, LCAS can achieve higher channel utilization compared to MMAC, by scheduling links and assigning different channels to different links, even within a flow.

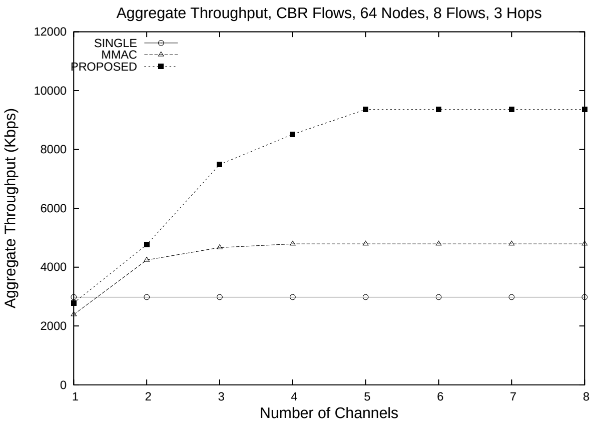
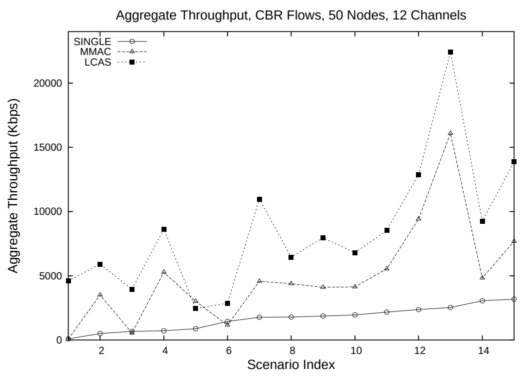


Figure 7.17: Aggregate throughput varying number of channels.

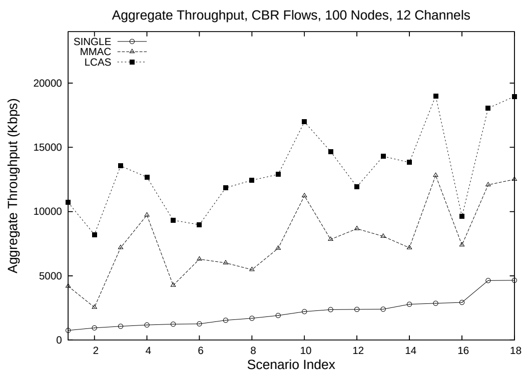
Random Topology

For the next set of simulations, we ran protocols a random topology. Nodes were placed randomly in the 1000m $\times$ 1000m square area, and source and destination nodes were also chosen randomly. In random topology simulations, we simulated FTP traffic (over TCP) as well as CBR traffic, in order to see how well LCAS supports TCP flows.

CBR traffic, 12 channels In the first simulation, we ran 10 flows of CBR traffic in random topology scenarios with 12 channels. 50 nodes and 100 nodes were placed in the simulation area to create a sparse and dense network. We measured the aggregate throughput with each source sending CBR flow at 4Mbps. The results are shown in 7.18. The result shows that LCAS achieves higher throughput compared to MMAC in all simulated scenarios. Depending on the scenario the improvement ranges from 20% to 100%. Considering node density, results from Figure 7.18(a) and 7.18(b) show that when the network is dense, both LCAS and MMAC gain benefit because flows can be less overlapping compared to a sparse network.



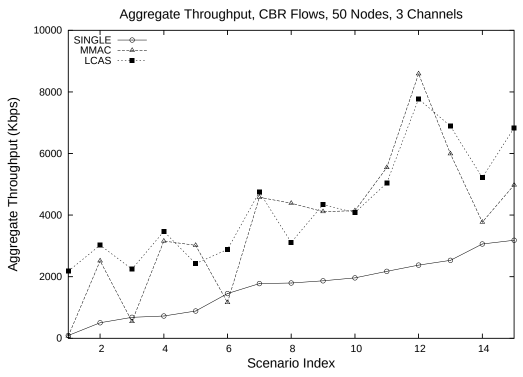
(a) 50 nodes



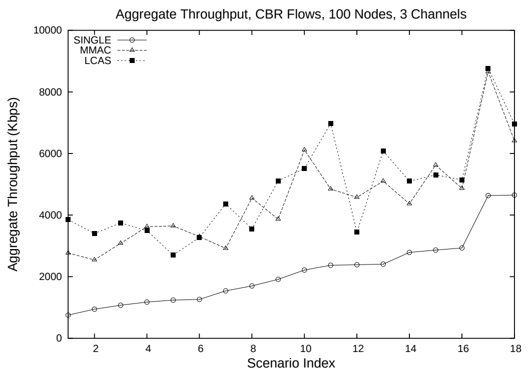
(b) 100 nodes

Figure 7.18: Aggregate throughput for scenarios with 12 channels and CBR traffic.

CBR traffic, 3 channels Next, we ran simulations with the same scenarios, but the number of channels was set to 3, instead of 12. The results are shown in Figure 7.19. The results show that, when there are small number of channels, LCAS and MMAC outperform each other depending on the scenario. In these scenarios with 3 channels, performance of these two protocols are comparable, and they both outperform the single-channel DCF.



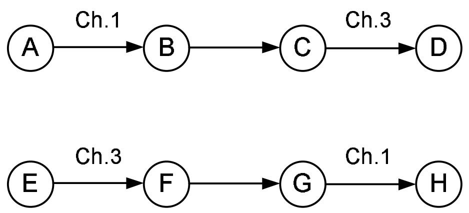
(a) 50 nodes



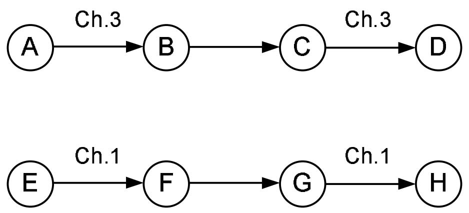
(b) 100 nodes

Figure 7.19: Aggregate throughput for scenarios with 3 channels and CBR traffic.

The reason that LCAS does not perform better than MMAC with small number of channels is as follows. The difference between MMAC and LCAS is that MMAC tends to assign a common channel for the nodes within a flow, where LCAS assigns different channels to links within a slot. Thus, LCAS can utilize channel diversity better than MMAC. However, when there are small number of channels, there is not much diversity that LCAS can benefit from. With 3 channels, MMAC already utilizes most of the bandwidth available, and thus the throughput of MMAC is comparable to that of LCAS.



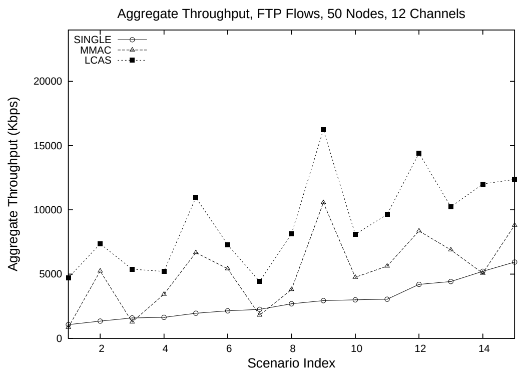
(a) LCAS



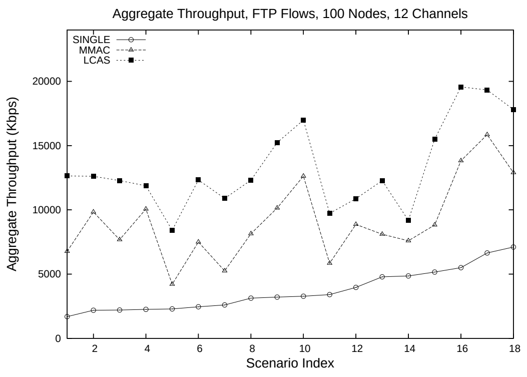
(b) MMAC

Figure 7.20: An example scenario illustrating channel assignment strategies of LCAS and MMAC.

FTP traffic, 12 channels Finally, we have simulated TCP traffic on the protocols. Since LCAS schedules time slots to links, the variation in packet delay increases. The TCP throughput reflects the impact of the variation in packet delay, as well as the impact of packet loss. In scenarios with 12 channels, 10 flows were created, and each flow runs FTP traffic over TCP. We have measured the aggregate throughput, and the results are shown in Figure 7.21. The results show that LCAS achieves higher throughput with TCP traffic in most cases, compared to MMAC and DCF.



(a) 50 nodes



(b) 100 nodes

Figure 7.21: Aggregate throughput for scenarios with 12 channels and FTP traffic.

Overall, LCAS successfully utilizes multiple channels with single-interface nodes to improve network performance. With channel scheduling, the protocol reduces contention within a flow and also contention among overlapping flows. As a result, LCAS gains benefit over single channel protocols and other multi-channel protocols. The benefit is gained for both UDP and TCP traffic.

7.4 Comparison of the proposed protocols

In Chapter 3, we proposed MMAC, a multi-channel MAC protocol. In Chapter 5, we introduced AODV-E, a modification to AODV that prefers non-overlapping routes. AODV-E helped MMAC to utilize more channel diversity, thus improving the network throughput. In Chapter 6, we proposed MCRP, a routing protocol that utilizes multiple channels. Finally, in this chapter we proposed LCAS, a link-layer channel assignment and scheduling algorithm.

In this section, we compare the performance of these protocols using simulations. Six schemes are compared: SINGLE, MMAC/AODV, MMAC/AODV-E, MCRP, LCAS/AODV, and LCAS/AODV-E. These names refer to the MAC and routing protocols used. For the case of SINGLE, IEEE 802.11 DCF is used as the MAC protocol, and AODV is used for the routing protocol. For MCRP, IEEE 802.11 DCF is used as the MAC protocol.

The simulated scenario is shown in Figure 7.22. The simulation area is a 600m $\times$ 600m region. In the simulation area, there is a inner region which is 500m $\times$ 500m. 20 nodes are placed in the outer region, and 30 nodes are placed in the inner region. Only nodes in the outer area can become source or destination nodes, and nodes in the inner area act as relay nodes. The region for placing source and destination nodes at the edge of the simulation area is to create multi-hop flows and avoid single-hop flows. Since the protocols are developed for single-interface environment, a node cannot access multiple channels simultaneously, and thus channel diversity cannot be exploited in a single-hop flow. So we use multi-hop flows for the purpose of comparing our proposed protocols. We use 12 channels, and the bit-rate is 11 Mbps for each channel. Transmission range is approximately 250 m. 8 source-destination pairs are chosen, and each source creates a CBR traffic of 4Mbps. We run simulations on 10 different topologies, and simulation time is 30 seconds in each run. Each data point is obtained by averaging results from 10 simulation runs. Note that although a different seed was used for each run, the network topology and the source-destination pairs were the same in all 10 runs.

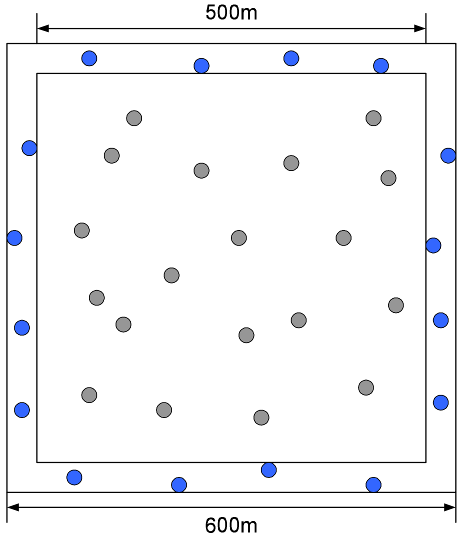


Figure 7.22: Simulation setup: nodes in the outer region can become source or destination, while nodes in the inner region are relay nodes.

The simulation result is shown in Figure 7.23. Standard deviation as a percentage of mean for each data point was around 5%, and was always less than 20%. The result shows that MMAC subtantially increases throughput over SINGLE by utilizing multiple channels. MCRP performs better than MMAC in average for most scenarios, and for some scenarios their performance are comparable. LCAS performs the best among the protocols by efficiently utilizing channel diversity.

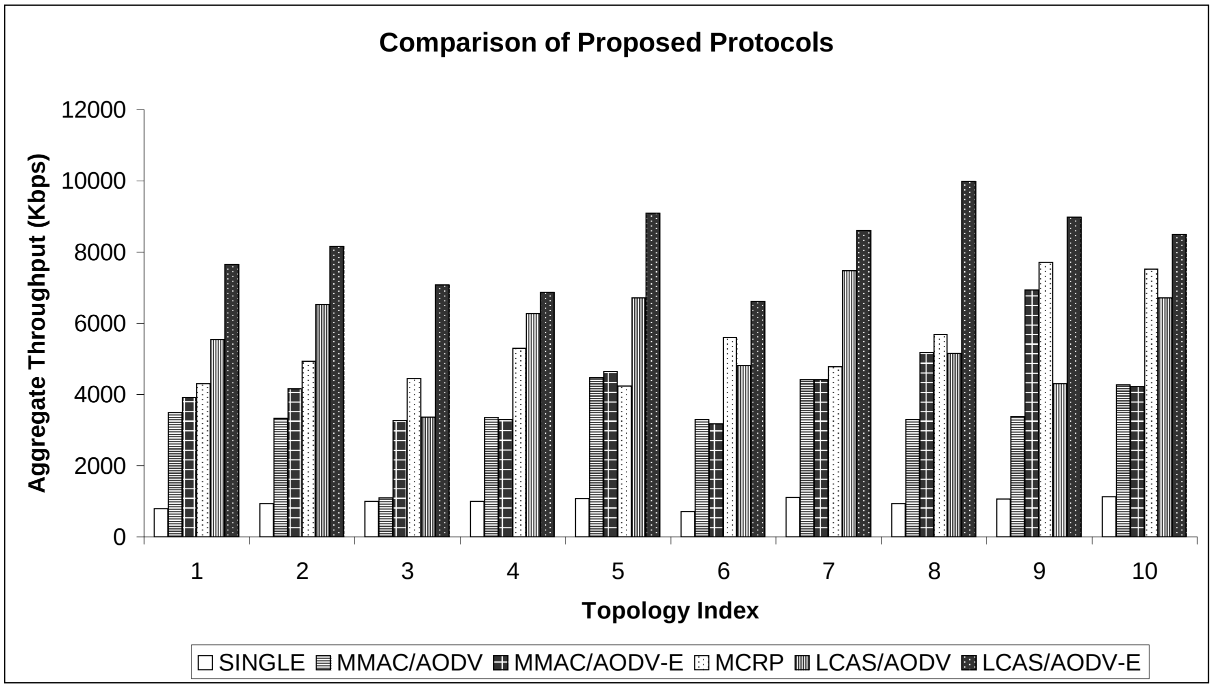


Figure 7.23: Simulation results: aggregate throughput of six different schemes.

When AODV-E is used, the throughput of MMAC and LCAS become higher compared to when AODV is used. We have already shown in Chapter 5 that AODV-E helps MMAC to utilize more channel diversity by preferring non-overlapping routes. The result shows that AODV-E can also help LCAS to improve throughput substantially. LCAS tries to assign different time slots to adjacent links so that multiple channels can be allocated for contiguous links. However, since a node only has a single interface, the limiation that only one link can be active at a time among all links adjacent to a node still exists. Thus, having non-overlapping routes improves the throughput of LCAS by allowing more simultaneous transmissions on different channels. For some topologies, the throughput of LCAS/AODV is less than the throughput of MCRP (Scenarios 3, 6, 8, 9, 10) and MMAC/AODV-E (topology 9). In these cases, routes are highly overlapping with LCAS/AODV and they limit LCAS from exploiting channel diversity. Whereas MMAC/AODV-E and MCRP find non-overlapping routes and thus utilize channel diversity better than LCAS/AODV. (MCRP tends to avoid overlapping routes by preferring “free node” in the route discovery. For details, refer

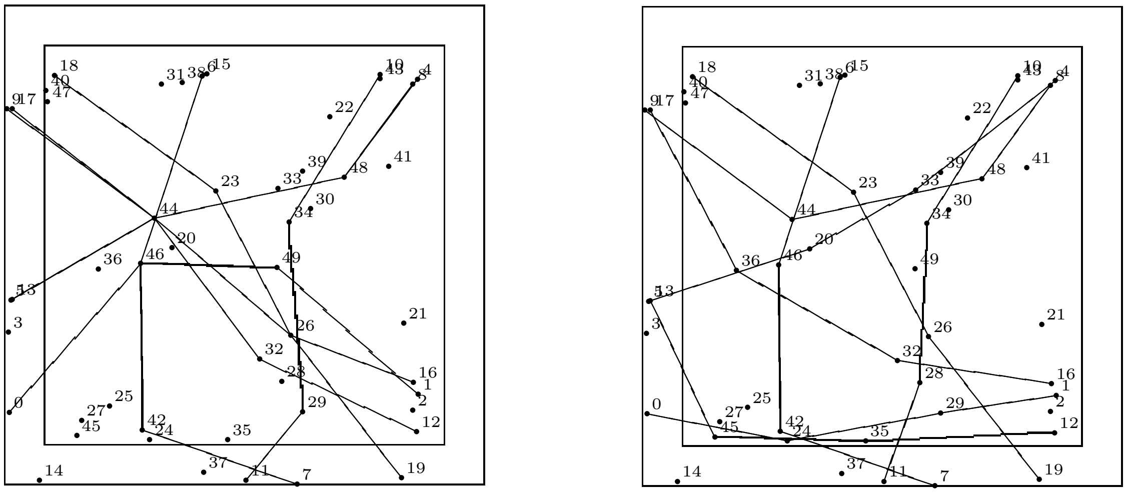


Figure 7.24: Network topology and selected routes for scenario 9 in Figure 7.23. The left figure shows the routes selected by AODV, and the right figure shows the routes selected by AODV-E. In the left figure, four routes are overlapping at node 44.

to Chapter 6.) For example, the network topology and selected routes for Scenario 9 are shown in Figure 7.24. The left figure shows the routes selected by AODV, and the right figure shows the routes selected by AODV-E. In the left figure, four routes are overlapping at node 44. Because of highly overlapping routes, the ability of LCAS to utilize channel diversity is limited. However, in the right figure, routes are more distributed among nodes. Thus LCAS can produce higher throughput by utilizing channel diversity better.

7.5 Summary

In this chapter, we proposed LCAS, a link-layer protocol for multi-channel networks that can support nodes with a single interface. The main benefit of LCAS is that it can assign different channels within a flow, thereby reducing self-contention and selecting channels based on the quality of channels that are spatially non-uniform. The channel scheduling in LCAS is coarse-grained, rather than per-packet basis, because frequent channel switching incurs high switching overhead. In addition to channel scheduling, a channel assignment protocol is proposed, which assigns channels to links based on channel usage in the neighborhood.

LCAS runs on top of any MAC protocol, such as IEEE 802.11 DCF. Thus, LCAS can be easily implemented by inserting a software layer. The simulation results show that LCAS improves network throughput significantly compared to IEEE 802.11 DCF running over a channel. Compared to MMAC, the performance of LCAS is comparable or superior, depending on the traffic load and number of available channels. LCAS improves upon MMAC especially when the number of channels is large, and flows are overlapping at nodes.

Although LCAS assumes that each node is equipped with a single interface, there may be nodes with multiple interfaces. In this case, the node can first assign links to interfaces, and then apply the channel scheduling algorithm for each interface separately. For example, if a node has two interfaces and four adjacent active links, the node can assign two links on one interface and other two links on the other interface, and then assign slots to the links in each interface. With interface assignment, LCAS can be extended to an environment where nodes are equipped with different number of interfaces.

Chapter 8

Multi-Channel Routing Protocols for Hybrid Networks

In the previous chapter, we have looked at multi-channel routing protocols for general ad hoc networks, where traffic source and destination can be arbitrary nodes. The routing protocol developed for ad hoc networks is somewhat complicated because the protocol needs to establish and maintain routes between any pair of nodes and also assign channels to those routes. A different architecture that has practical use is hybrid networks, which is a combination of infrastructure and ad hoc architectures. In this architecture, all traffic goes through access points, and all mobile nodes need to maintain at least one route to an access point. (An example of hybrid networks is discussed in [17]. Thus, routing protocols for hybrid networks can be much simpler compared to general multi-hop networks.

In this chapter, we develop routing protocols for hybrid networks. First, we define hybrid networks and discuss issues in developing multi-channel routing protocols for this architecture. Then, we develop and evaluate a routing protocol for hybrid networks, which balances load among channels by having each node select channels based on current traffic load.

8.1 Hybrid Networks

A multi-channel multi-hop wireless network of interest in this chapter can be considered an extension to infrastructure networks, allowing nodes to connect with an access point via multiple hops. An example network is illustrated in Figure 8.1. In the figure, solid lines indicate links on channel 1, and dashed lines indicate links on channel 2. The dotted line indicates that there is a potential link between D and E, if their channels match each other.

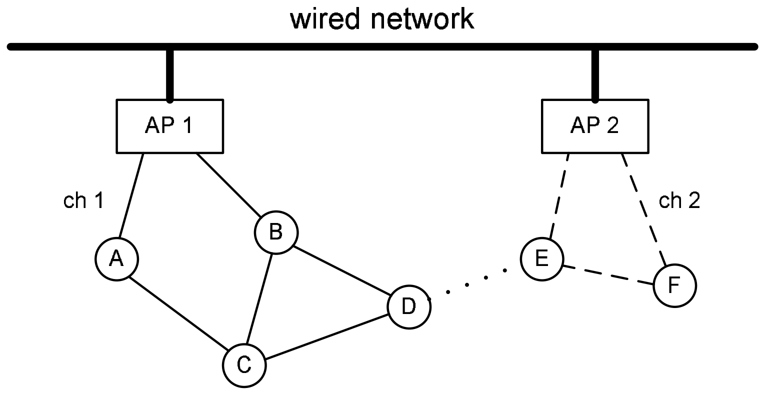


Figure 8.1: An illustration of a multi-channel multi-hop wireless network. Solid lines are links on channel 1, and dashed lines are links on channel 2. The dotted line indicates a potential link, if node D and E were on the same channel.

In this example, nodes A, B, C, and D are associated with AP1 on channel 1, and nodes E and F are associated with AP2 on channel 2. Nodes C and D cannot reach an access point directly, but they are connected via multiple wireless hops.

Note that a "node" can be a mobile host or a wireless router. Mobile hosts are end-user devices, and wireless routers are simple routers with only wireless interfaces, and they act as intermediate nodes to relay packets. Wireless routers are always willing to relay packets, whereas mobile hosts may or may not volunteer to relay packets of other mobile hosts. In the proposed protocol described in Section 8.2, mobile hosts that are not willing to relay packets of other hosts do not participate in the protocol to send HELLO messages or reply to SCAN messages (details explained later).

Coming back to Figure 8.1, consider node D. It is currently on channel 1, and is associated with AP1. However, if D switches its channel to channel 2, it can associate with AP2 via node E. Once D associates with AP2, node B and C can also connect to AP2 via D and E.

Since node D has two potential routes it can use, it must choose the route where it can achieve a better quality of service. The quality of service at a node including current traffic load on the channel and the quality of links on the route affected by environmental factors. Here, we mainly focus on the traffic load when selecting routes. Considering link quality as a factor in load metric can improve the accuracy of the metric.

Node D chooses the route with less traffic load. In order to do that, D must know the load on its current channel as well as other channels. Thus, we discuss how to estimate traffic load in the following subsection.

8.1.1 Estimating traffic load

Before discussing how to estimate traffic load, we state our assumptions. First, although a node may have multiple routes to the access point, only one route is used at any given time, and other routes are maintained for backup so that they can be used when the primary route fails or becomes congested. For example, in Figure 8.1, node D only uses the route through node B to connect to the wired network (this route is called the *primary route*. The primary routes of nodes associated with the same AP form a *route tree*, rooted at the AP. Second, we assume that most of the traffic is downlink traffic (e.g. accessing web data), sent from AP to mobile hosts. The proposed protocol supports uplink traffic, but the load estimation is based on the downlink traffic. Third, we assume that APs are placed dense enough that most routes are short in terms of number of hops, such as 3 or 4 hops (similar assumptions are made in other works [12, 59]). Thus, there is little chance for simultaneous transmissions within a route tree. When the depth of a route tree becomes large, spatial reuse must be taken into account in the load estimation. Finally, we assume that as in single-hop infrastructure networks, neighboring APs are typically assigned different channels. So it is unlikely that a node finds short routes to two different APs that are on the same channel. Thus, we convert channel load balancing problem into balancing load among route trees.

To discuss how to estimate traffic load, we refer to Figure 8.1 again. Currently node D is connected to AP1 via node B (using channel 1). D has another route to AP2 via node E, but it is not used presently. Suppose each node exchanges its traffic load information via control messages (the protocol details are explained later). So D obtains load information from B, C, and E. What would be the metric that nodes should use to communicate the load information? First, each node can measure the number of bytes it has received or forwarded during a recent time window. For example, during last 10 seconds, the average traffic load that node B has received or forwarded traffic is 500 Kbps, and the average traffic load that node E has received or forwarded is 100 Kbps. Does this information suggest that the load on channel 2 is lighter than the load on channel 1, so that it is better for node D to switch to channel 2 and join AP2 route tree? The answer is no. Even if E is only receiving 100Kbps, it does not mean that the load on channel 2 is 100Kbps. AP2 might be sending 1Mbps of traffic to node F. Thus, locally measured load cannot be used as basis for selecting routes.

Another metric that can be used is the load measured at the AP. Since all the traffic destined to the nodes associated with the AP goes through the AP, it can accurately measure the load on its route tree. We assume that the bandwidth of the wired backbone that the APs are connected to is much larger than the bandwidth of wireless links. Suppose AP1 observes that during last 10 seconds, it has forwarded 2Mbps of traffic. Also, AP2 has forwarded 1Mbps of traffic. If D obtains this information, D knows that AP2 has a lighter load than AP1. However, the AP-measured load is still not an accurate measure that can be used in selecting routes. Consider the scenario in Figure 8.2. Currently, D is associated with AP1 on channel 1, via node B. Suppose AP1 has 2Mbps of traffic destined for node A, and AP2 has 1Mbps of traffic destined for node F. If node D obtains this information, does this suggest that node D should switch to channel 2 and connect to AP2? The answer is no. Since each packet needs to be transmitted three times to reach node F, the actual load on the route tree is 3Mbps instead of 1Mbps (recall that due to small depth of the tree, two transmissions in the same route tree are assumed to interfere with each other). So it is better for D to stay on channel 1.

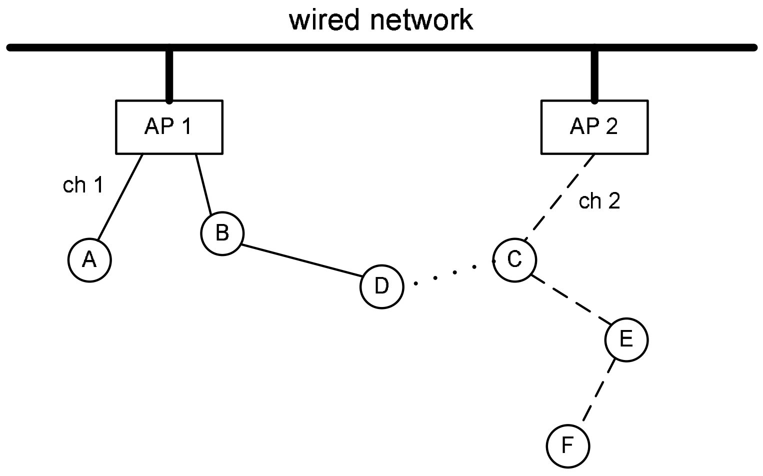


Figure 8.2: An example network scenario. This example indicates that number of hops must be considered in measuring the load. The solid lines are links on channel 1, and the dashed lines are links on channel 2.

This example indicates that the load should be weighted according to the number of hops to the destination. We call this new metric the *weighted-load* metric, and we use this metric for load measurement. The specific details of how the load is measured at the AP and how the load information is distributed is explained in Section 8.2. Next we discuss how a node should select routes based on this load information.

8.1.2 Selecting the route with minimum load

Suppose a node obtains load information on all its potential routes to destinations. When does a node decide to switch channels and join another route tree? This subsection discusses this issue. A node cannot freely switch channels because it might have child nodes in the route tree. Consider the scenario in Figure 8.3. Initially node D is associated with AP1, and so is node G. Suppose AP1 has 1Mbps of traffic for node A, 1Mbps for node D and 1Mbps for node G. Also, AP2 has 1Mbps for node F. If node D obtains this information, should node D switch to channel 2?

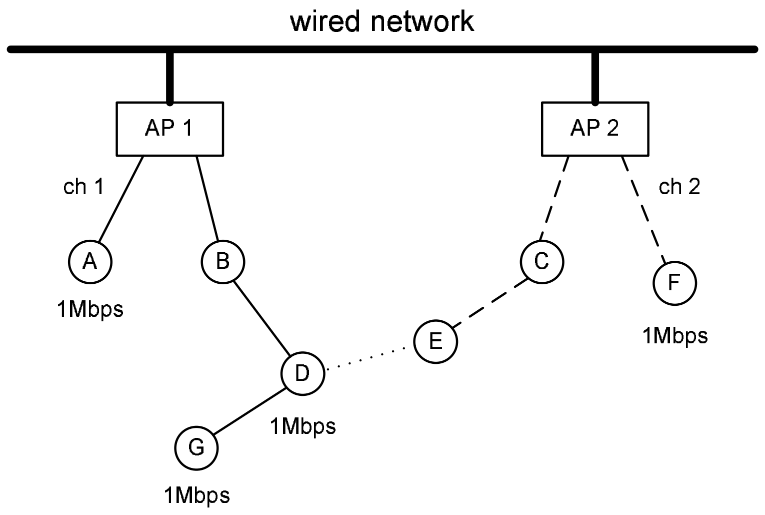


Figure 8.3: An example network scenario. This example indicates that subtree load must be considered when selecting the best route.

Using the weighted load metric, the load of AP1-tree (the route tree rooted at AP1) is 6 Mbps (1 Mbps for A, 2 Mbps for D, and 3 Mbps for G), and the load of AP2-tree is 1Mbps. If only node D can switch to channel 1, the load of AP1-tree will become 4 Mbps, and the load of AP2-tree will become 4 Mbps (1 Mbps for G and 3 Mbps for D). So this suggests that D should switch to channel 2. However, it will lead to node G being disconnected from the network. So when D decides to switch channels, all its descendants in the route tree must also switch channels (node G might have another route to the AP on channel 1, but D does not know this). But if D and G switch together, load of AP2-tree becomes 8Mbps, and thus D may decide to stay on channel 1.

This example indicates that a node D decides to move from AP1-tree to AP2-tree only when the current load of AP1-tree is larger than the current load of AP2-tree plus the load of the subtree rooted at node D weighted according to the number of hops in the AP2-tree. If the current load and the load after D moves is equal, tie is broken using number of hops from D to the APs.

A node may decide to switch its primary route within the tree (i.e., without switching channels or associating with another AP). This happens when the primary route has larger number of hops from the AP than the alternative route. Then the weighted load after the node switches its primary route will be smaller than the current load. So the weighted load metric prefers routes with smaller hops. A detailed description of how a node selects its primary route is presented in Section 8.2.

8.2 Proposed Routing and Channel Assignment Protocol [2]

In this section, we describe our routing and channel assignment protocol in detail. As mentioned earlier, we assume that all nodes in the network communicate via access points, and not with each other directly. Whenever two mobile nodes need to communicate, they can use their routes through APs. So it is enough that each node maintains at least one route to an access point, and routes to all the descendant nodes in the route tree. The AP must maintain routes to all the nodes associated with the AP.

The routing protocol must answer the following questions:

- How are the routes established?
- How are the routes maintained and updated?
- How are the routes recovered after failures?

In the following subsections, we describe how the proposed protocol addresses these issues.

8.2.1 Route establishment

When a node is turned on, it must first discover a route to an access point. For this purpose, the node performs an *"active scanning" on all channels.* Consider the scenario in Figure 8.4. There are two APs, operating on channel 1 and channel 2, respectively. Before node B joins the network, node A is already in the network, associated with AP1 on channel 1 (as shown in Figure 8.4(a). Now node B joins the network as in Figure 8.4(b). Initially, node B selects a random channel, and starts scanning by broadcasting a SCAN message on the channel. After sending the SCAN message, node B waits on the channel for some time to collect responses and then moves on to the next channel and eventually scans all channels in a round-robin manner.

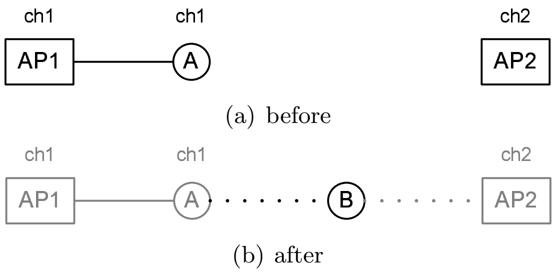


Figure 8.4: A simple network scenario with two access points and two nodes.

Access points, and nodes that are already associated with an access point can reply to the SCAN message by unicasting a REPLY message back to the sender, node B in our example. The REPLY message contains the address of the replier, the address of the AP that the replier is associate with, the load on the channel, and the number of hops to the AP. In the above scenario, node A replies to SCAN on channel 1, and AP2 replies on channel 2. Since there can be multiple neighbors replying on a channel, nodes wait for a random delay before sending the REPLY message.

After scanning all channels, node B selects its *primary route* by choosing one of its neighbor as its *parent* node. Among all the routes received, B selects the route with the minimum load according to the weighted-load metric explained in Section 8.1. If there is a tie, the one with the minimum number of hops is chosen. In the above example, if both channel 1 and channel 2 do not have any load, B selects AP2 as its parent node since because the hop distance is smaller. Once a node selects its primary route, the path from the node to the AP is established. Then, node B sends an ASSOCIATION message using the selected route, so that a reverse path is set up from the AP to node B.

The route table that each node maintains is similar to that of AODV [18], with the following changes.

- There are new fields in the route entry. The "type" field indicates the node type of the destination. If the type is "MH", destination is a mobile node. If the type is "AP", it is an access point. If the type is "PRIM", the destination is an access point, and it is the primary AP that the mobile node is currently associated with. The "chan" field indicate which channel to use for the route. The "load" field shows the load information, which will be explained later.

- For uplink routes (routes to from a mobile host to an access point), the entire path information is recorded in the route entry. (The approach of keeping the entire path information is also used in [70].) Path information is not maintained for downlink routes.

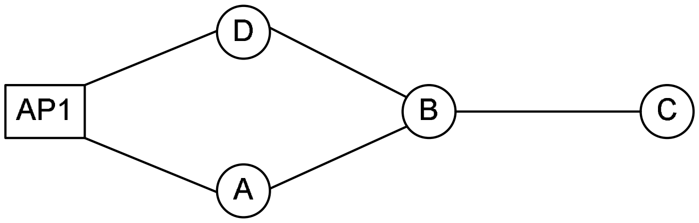
- A mobile node may keep multiple routes to a single access point. When the primary route is broken, other routes can be used as backup routes.

An example network topology and route table is shown in Figure 8.5.

In the topology shown in Figure 8.5(a), node B has two routes to AP1, and a route to node C. Between the two routes to the AP, node B has chosen the route via node A as its primary route.

8.2.2 Route management and updates

Managing and updating routes is the most important part of our proposed protocol. Once the primary route has been established, each node collects load information for its own route tree and other route trees. Based on this information, the node may switch to the route tree with minimum load so that it can obtain the highest quality of service possible. First, we describe how the load is measured at the APs. Next, we explain how the load information is collected by the nodes. Finally, we present the process of route update.



(a) Network topology

Route Table of B

dst	nexthop	hops	type	chan	load	path
AP1	A	2	PRIM	1	500	A AP1
AP1	D	2	AP	1	500	D AP1
C	C	1	MH	1	0	

(b) The route table of node B

Figure 8.5: An example route table and its corresponding network topology. Node B has two routes to AP1, and a route to node C. Between the two routes to AP1, the route via node A is selected as the primary route.

Measuring load

In Section 8.1, we have suggested to use the *weighted load* was the suitable measure. Here we present the detailed description of how the load information is collected and distributed. Note that the protocol performs load balancing based on the downlink traffic, because we assume that the downlink traffic is much more dominant than uplink traffic. Although not considered in estimating load, the protocol supports uplink traffic as well as downlink traffic.

Each AP remembers the amount of downlink traffic it has seen during past T seconds. In the simulations, we have used 10 second as T. The packets counted as traffic are the ones that are from wired network to a node in the route tree rooted at the AP. Since the AP knows the destination, it records the amount of traffic per destination.

For example, let us consider Figure 8.5(a) again. Suppose during last T seconds, AP1 has received 100Kbps of load for node D, and 200 Kbps of load for node B. The AP1 records this information in its route table as in Figure 8.6.

The weighted load metric indicates that the load for each destination should be weighted by

Route Table of AP1

dst	nexthop	hops	type	chan	load	path
A	A	1	MH	1	0	
B	A	2	MH	1	200	
C	A	3	MH	1	0	
D	D	1	MH	1	100	

Figure 8.6: Route table of AP 1 in the scenario shown in 8.5(a).

the number of hops from AP to the destination node. So the weighted load of the route tree L_1 is computed as follows.

$$L_1 = \sum_i (h_i \times l_i) \tag{8.1}$$

where i is a node in the route tree rooted at the AP, h is the number of hops, from AP to the node, and l is the amount of traffic destined for the node. So in the above example, the total load of AP1-route tree is 500Kbps.

Distributing and collecting load information

How a node makes decision on which route tree to stay on was explained in Section 8.1. To make the decision, a node should obtain load information on its own route tree, other route trees and the amount of traffic destined for the node itself and its subtree.

To allow each node to obtain the load information of its subtree, the AP piggybacks the load information in the data packets. For example, in scenario shown in Figure 8.5(a), AP1 observes that 200Kbps of traffic has been received to be delivered to node B during last T seconds. Then AP1 sends the load information (200 Kbps) with the data packets along the route to B. The intermediate nodes and the destination node record the information in their route table. So in the example, node A records 200 Kbps in the route entry that has node B as destination, and node B records 200 Kbps as a separate variable name "LOAD" in its route table. Note that piggybacking load information in every packet may be a high overhead. In order to reduce the overhead. In order to reduce the overhead, an AP can piggyback load information in every k packets. (In the simulations, we piggyback load information in every packet.)

Now a node has to obtain information on route trees. Periodically, each AP transmits a HELLO message, which includes the identifier of the AP and the aggregate load information measured using weighted-load metric. Similar to the scanning process, HELLO messages are sent on all channels, one at a time. When nodes receive the HELLO message, they update their route table according to the information given in the message (as explained later). After that, only if the sender of the HELLO message is the *next hop* node in its primary route, the node forwards the HELLO message. Otherwise the packet is discarded. To avoid collision among nodes that transmit HELLO messages at the same time, each node waits for a short random delay before sending its HELLO message. In this manner, the HELLO messages are initiated by the APs and forwarded along the route tree.

We call the period for sending HELLO messages P_{hello}. P_{hello} must be long enough to reduce overhead on the network. In the simulations, we have used 3 seconds as the P_{hello}. To avoid synchronized HELLO period among APs, each AP randomly picks the next HELLO time between the range [1.5-4.5] seconds.

Since the HELLO messages are sent on all channels, a node can receive HELLO from all the neighbors including those on other route trees. When a node receives a HELLO message, it first checks whether a new route can be established through the sender of the HELLO message. If so, then the new route is recorded in the node's route table. Then the node updates load information for the route tree that the sender is on. Consider the following scenario in Figure 8.7.

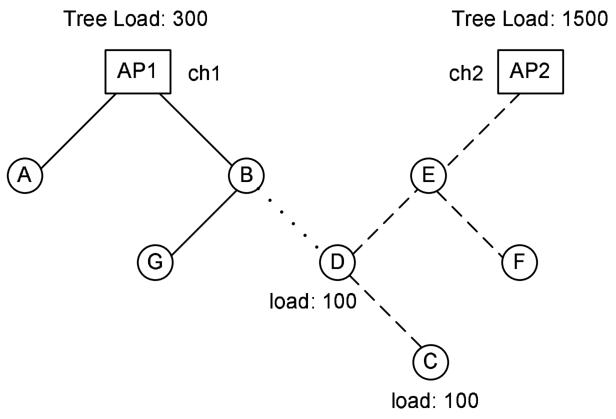


Figure 8.7: An example network scenario to illustrate the process of obtaining route information and selecting primary route based on load information.

In Figure 8.7, node D is initially associated with AP2 on channel 2. AP2 has observed that 100 Kbps of load is for node D and 100 Kbps of load is for node C. As the data packets are forwarded, D obtains load information of itself and node C. When AP2 broadcasts a HELLO message, D learns that the load of its route tree is 1500 Kbps. Now at some point of time AP1 starts HELLO process. Node B receives the HELLO message and rebroadcasts it on all channels. When B transmits the HELLO packet on channel 2, node D receives the packet. Now D finds out that node B is associated with AP1, and is 1 hop away from AP1. So D obtains a backup route to AP1 on channel 1, through B. In the HELLO message, B includes the load of its route tree, which is 300Kbps. So after receiving the HELLO packet and updating its route table, the route table of node D looks like Figure 8.8.

Route Table of D Load: 100

dst	nexthop	hops	type	chan	load	path
AP2	E	2	PRIM	2	1500	E AP2
C	C	1	MH	2	100	
AP1	B	2	AP	1	300	B AP1

Figure 8.8: Route table of node D in the scenario shown in 8.7.

Using this information, node D can now decide if it should switch to the other route tree.

Switching route trees for load balancing

Once the necessary load information is obtained, nodes can decide whether to switch to other route trees. In the example shown in Figure 8.7, node D can switch its channel to channel 2 and re-associate with AP1, because it has a lower load. When making the decision, the node compares the current load of its route tree and the load of the other tree *when the node joins the tree.*

Since node D has children in the route tree, it cannot just switch channels to join other trees, because the child nodes will lose connections with the AP. Instead, if D decides to switch channels, it should tell all its children to switch channels as well. Effectively, the whole subtree moves to the new route tree. So the load information should be computed correspondingly.

For example, in Figure 8.7, suppose node D wants to decide if it should move to AP1. The current load of AP2-tree is 1500Kbps. Now the load of the other tree should be computed as $L_{AP1'} = L_{AP1} + \sum_i (h_{iAP1} \times l_i)$.

where $L_{AP1'}$ is the load of AP1-tree after node D joins the tree, L_{AP1} is the load of AP1-tree before node D joins the tree, i is a node in the subtree rooted at node D, h_{iAP1} is the number of hops from node i to AP1, and l_i is the load destined for node i. In the above example, the load of AP1-tree after the subtree of D joins the tree is computed as $L_{AP1'} = 300 + (100 \times 2 + 100 \times 3) = 800$

Since it is still smaller than the current load of AP2-tree, node D can decide to switch channels so that it can join AP1.

Even if node D observes that AP1 has less traffic load than AP2, it does not immediately move to AP1, because the decision can be based on out-of-date information. Also, reacting immediately can cause route oscillations, because multiple nodes can switch back and forth causing the traffic load to oscillate between two route trees. Instead, if node D observes that AP1 has lower load for sufficiently long time, it decides to switch channel with confidence that it will balance the load among APs. The duration of time a node waits before it switches route trees is a tunable parameter. We denote it as T_{switch} and we use $T_{switch} = 10$ seconds in the simulations.

Once node D decides to switch channels, it first sends a SWITCH message to all its child nodes, and the SWITCH message includes the new AP, number of hops from node D to the AP, and the new channel. The SWITCH packet is forwarded down the tree, and all children of node D switches their channels and update their route entry for the primary route. After sending the SWITCH packet, node D associates with the new AP by sending ASSOCIATION message on the new route. The ASSOCIATION message includes the previously associated AP, which is AP2 in this case. When AP1 receives the ASSOCIATION message, it informs AP2 through wired backbone network that node D has left AP2. All children of node D go through the same process to associate with the new AP.

8.3 Performance Evaluation

We have performed simulations to evaluate the performance of the proposed protocol. In this section, we report and discuss the results.

There are two main design goals for the proposed protocol. First, the protocol should allow every node in the network to find a route to at least one AP, if such a route exists. To avoid fast channel switching, we consider a route as valid only if all nodes on the path are on the same channel. Second, within the constraint that every node should have at least one route to an AP, the routing protocol should adapt to changing traffic conditions on channels and balance load among them to maximize channel utilization.

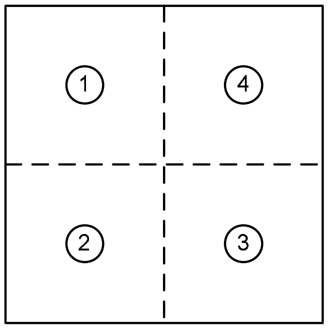
To see how well the proposed protocol utilizes available bandwidth in available channels, we compare our protocol with two other protocols. The first one is AODV [18], which is a single channel protocol. In AODV, access points and mobile nodes are not differentiated, and source node does route discovery in order to find destinations. When running AODV, all nodes including APs are assigned the same channel. The second one is a multi-channel protocol, but without load balancing. So number of hops is the only metric used to find the best route. We call this protocol "MCP" (Multi-Channel Protocol). With MCP, APs in a crowded area will have a correspondingly large number of nodes in its route tree, and APs in other areas will have small number of nodes associated with them. We call our proposed protocol as "MCP-LB" (Multi-Channel Protocol with Load Balancing), to distinguish from the other two protocols. In the following, we first describe our simulation setup, and then report the results.

8.3.1 Simulation Setup

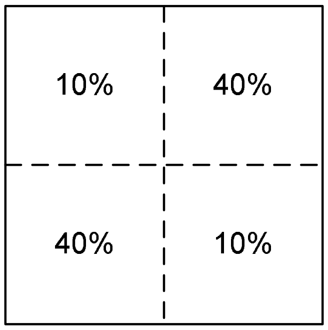
We have used ns-2 simulator [33] with wireless extensions for our simulations. The simulation area is a 1000m $\times$ 1000m square, where 64 nodes are randomly placed. The transmission range of each node is 250 m, and the channel bit rate is 2 Mbps. Each node uses IEEE 802.11 DCF for medium access control.

Unless otherwise specified, 4 APs are placed at the center of 4 quadrants, as in Figure 8.9(a). There are 4 orthogonal channels, and each AP operates on a different channel. Among 64 nodes in the area, 16 nodes are randomly picked as destination nodes that receive traffic from the wired network. Constant bit rate (CBR) traffic comes from the wired network through the AP and the AP forwards the traffic to the destination node. The size of each packet is 512 bytes. To create unbalanced traffic load in the area, we have picked the destination nodes using the distribution shown in Figure 8.9(b).

For the protocol parameters, we have set duration T, which is the duration of time window used by an AP to measure the load, as 10 seconds. Also, we have set the HELLO period, P_{hello}, to



(a) Placement of APs



(b) Placement of destination nodes

Figure 8.9: Placement of APs and destination nodes.

be 3 seconds and the minimum amount of time, T_{switch}, to be 10 seconds. These parameters were explained in Section 8.2.

Finally, the simulation time for each simulation is 400 seconds, and each data point in the graphs are a result of 10 runs, except for Figure 8.14 and Figure 8.15.

8.3.2 Results

In the first simulation, we measured the aggregate throughput of the three protocols, varying the total network traffic. The total traffic load is divided equally among flows. So if the total load is 4Mbps, the rate of each flow is 250Kbps. The results are shown in Figure 8.10.

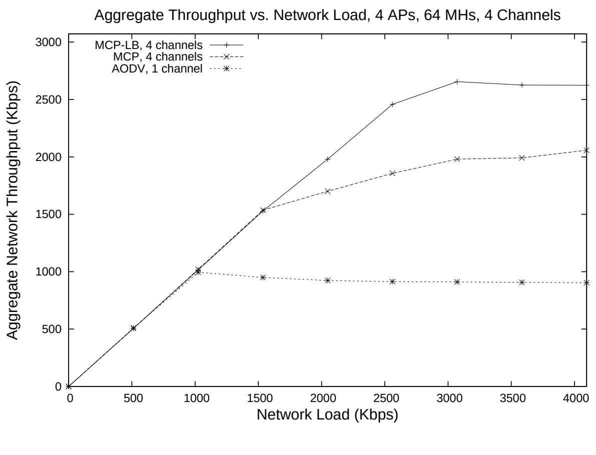


Figure 8.10: Aggregate Network Throughput Varying Network Load.

As shown in the graph, the throughput of the AODV is limited to around 1000 Kbps. For MCP and MCP-LB, MCP-LB achieves higher throughput than MCP. This is because the two channels that the APs in the crowded area use are congested, whereas the other two channels are under-utilized. Note that in the MCP, nodes select routes based on number of hops. Due to the spatially unbalanced distribution of the destination nodes (the selection of nodes is shown in Figure 8.9(b)), 80% of the nodes are associated with two APs in the crowded area, and only 20% of the nodes are associated with the other two APs. In MCP-LB, some destination nodes join route trees of the APs in the low-density area. Since these nodes are connected via multiple hops, the actual throughput achieved is less than the maximum achievable throughput, which is 4Mbps (the total load).

In the next simulation, we varied the number of channels to study its impact on the performance of the MCP-LB, the proposed protocol. For all scenarios, the number of APs is set to 8, and all APs are placed around the center of the area. For scenarios with channels less than 8, there are multiple APs on the same channel. 32 flows were generated for 32 different destination nodes. The result is shown in Figure 8.11.

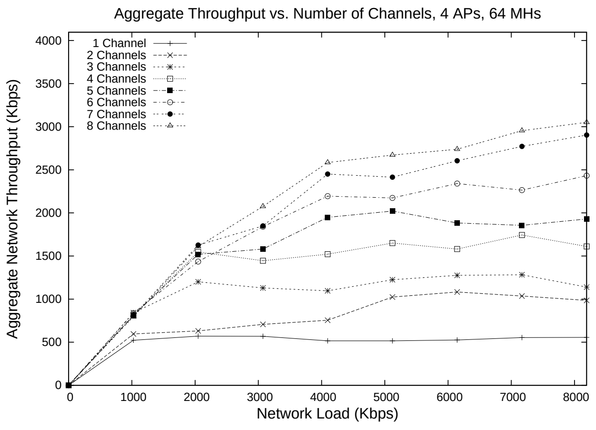


Figure 8.11: Aggregate Network Throughput Varying Number of Channels.

As shown in the graph, the throughput increases as the number of channels increase, almost in a linear fashion. We can observe that even with 8 channels, the achieved throughput is around 3Mbps, when the total network load is 8Mbps. This is because the average number of hops from source to the AP is approximately 2 hops. If the APs are placed in uniform distribution, the achieved throughput would further increase.

This observation leads us to our next simulation. We have simulated three scenarios with exact same setting, including the placement of mobile nodes and selection of destination nodes. The only difference between three scenarios is the placement of APs. In the first scenario, the 4 APs were placed in the center. In the second scenario, the APs were placed in the center of 4 quadrants, as in Figure 8.9(a). In the third scenario, the APs were placed at the 4 corners of the simulation area. The result is shown in Figure 8.12.

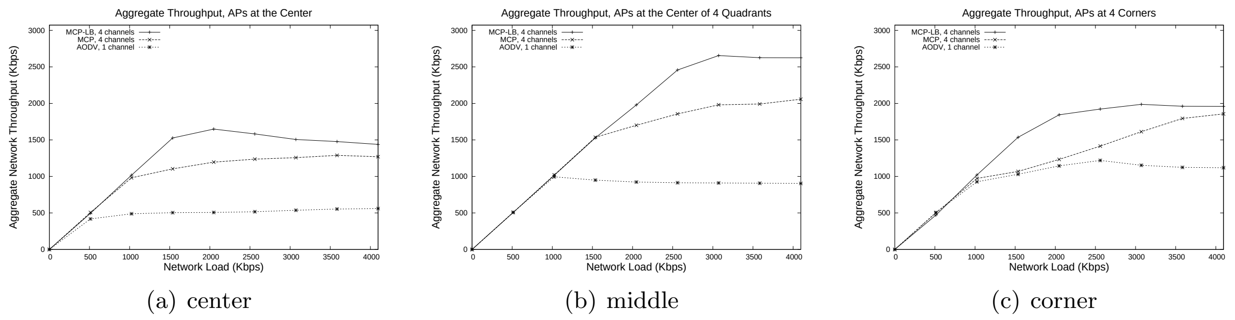


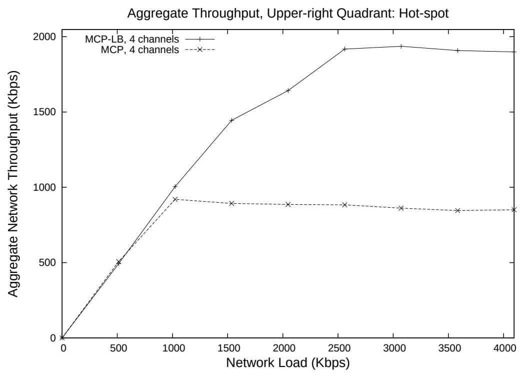
Figure 8.12: Aggregate Throughput varying Placement of APs.

Among the three scenarios, the throughput of MCP and MCP-LB are the highest in the second scenario, where the APs are placed in the center of 4 quadrants. This is because in the second scenario, the average number of hops is smaller, and the number of hops do not increase much even when a node switches to associate with an AP in another quadrant. When the APs are at the 4 corners, the benefit of load balancing is decreased because when nodes move to other route trees, the number of hops in the tree is very large so that the amount of throughput improvement is lowered.

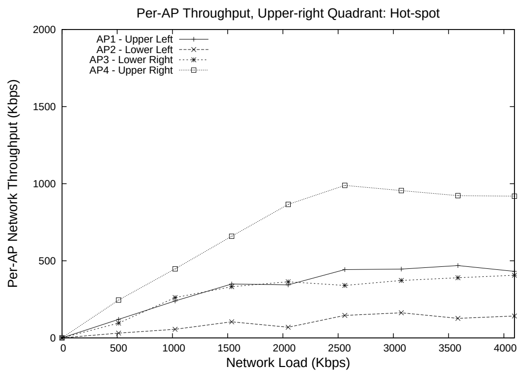
This result indicates that the density of APs is critical in the performance of our proposed protocol. If the APs are placed too far away so that a node has to use 5 or 6 hops to connect with another AP, the benefit of load balancing is reduced. To achieve significant benefit from load balancing, the APs have to be placed dense enough so that a node can find multiple APs in the range of 2 or 3 hops.

The next simulation is performed to see the performance of the proposed protocol in relieving the congestion in hot-spots by redirecting nodes to other APs. To make an extreme hot-spot, all destination nodes were selected from nodes in the upper-right quadrant. In Figure 8.13(a), the aggregate throughput of MCP and MCP-LB are shown. Since nodes associate with closest AP in the MCP, only one channel is used and other three channels are wasted. So the throughput is limited at 1Mbps. However, MCP-LB redirects nodes to other APs to improve performance. Figure

8.13(b) shows the per-AP throughput for MCP-LB. AP4, which is placed in upper-right quadrant where all destination nodes are placed, achieves a throughput of 1Mbps, because all destination nodes are in one-hop range of the AP. For AP1 and AP3, the throughput is around 40% of AP4. This indicates that the average number of hops the nodes connecting to these APs is approximately 2.5. Finally the throughput of AP2 is the least among APs. Since AP2 is placed far away from AP4, nodes have to travel approximately 5 hops to communicate with AP2. Although the throughput of the APs is different, the proposed protocol regards this as balanced, because it uses the weighted load metric, multiplying number of hops to the actual load for a node.



(a) Aggregate throughput



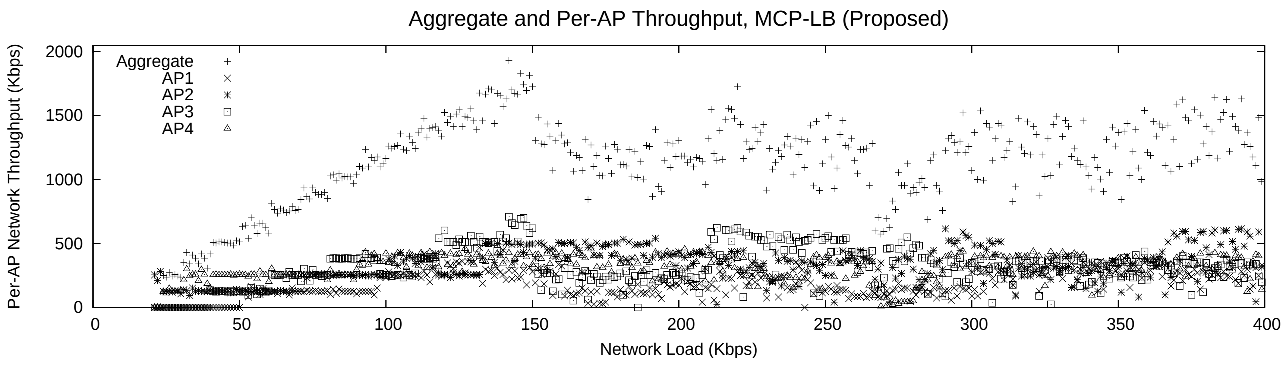
(b) Per-AP throughput

Figure 8.13: Aggregate and Per-AP throughput for the scenario with a hot-spot.

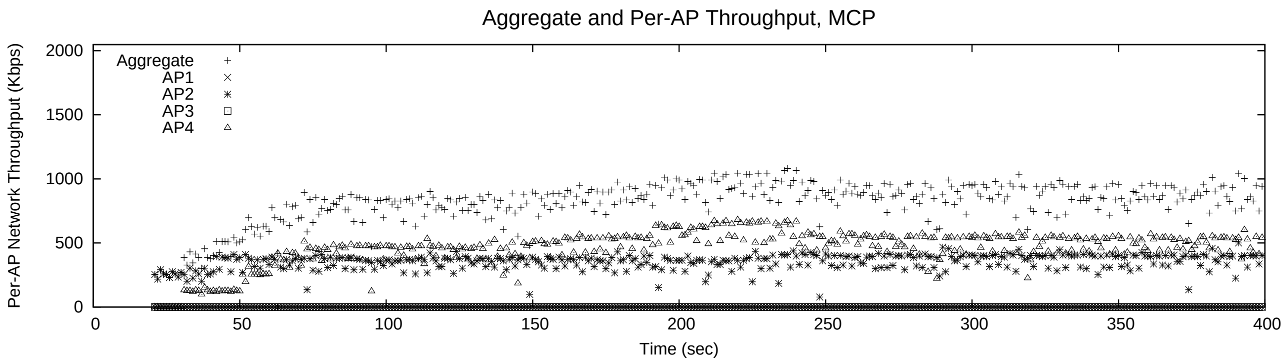
In our final simulation, we studied the adaptive behavior of our proposed protocol. During 400 seconds of simulation time, we simulated 32 flows, one flow starting at every 10 seconds. We plotted aggregate and per-AP throughput for MCP-LB and MCP. To create hot-spots, destination nodes were only selected from upper-right and lower-left quadrant. The result is shown in Figure 8.14. Comparing the two protocols, we can see that the proposed protocol utilizes all 4 APs by redirecting nodes to other APs, whereas with MCP, throughput of two APs are kept at zero. As a result, MCP-LB achieves significantly higher throughput than MCP.

In addition to the aggregate and per-AP throughput, we also plotted the weighted load at each AP to see how the proposed protocol balances the load among APs using the weighted load metric. The result is shown in Figure 8.15. This graph shows how the proposed protocol tries to balance the weighted load among APs, in the changing traffic conditions.

In conclusion, the proposed protocol successfully utilizes available bandwidth in available chan-



(a) MCP-LB (Proposed), 4 channels



(b) MCP, 4 channels

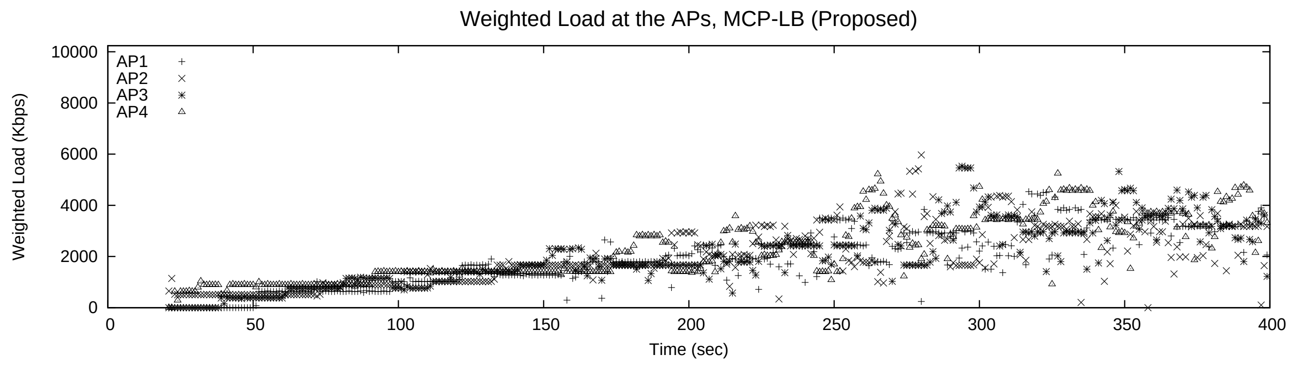
Figure 8.14: Aggregate and per-AP throughput.

nels, by balancing the load among APs. So it achieves significant improvements over MCP, when the traffic load is unbalanced in the area.

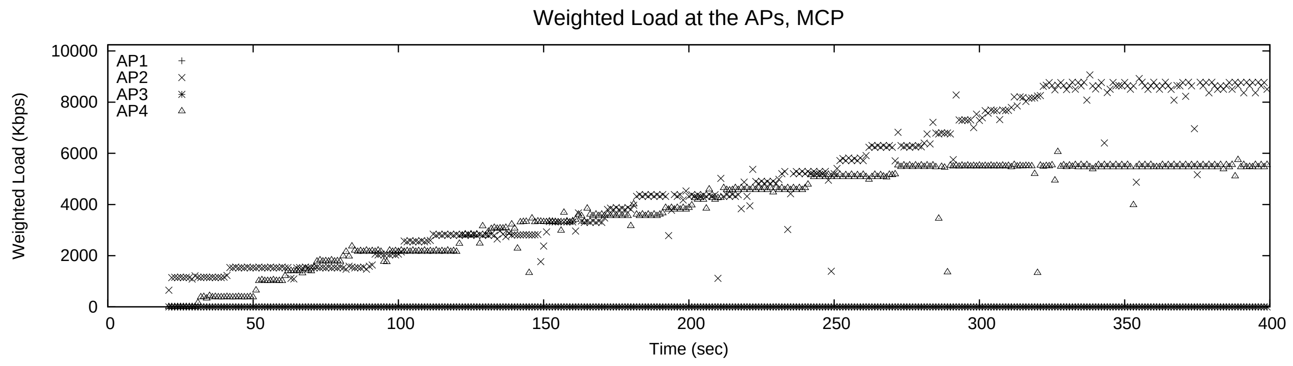
8.4 Related Work

There has been vast amount of effort in the research community to improve performance of wireless networks. One research direction that has gained increasing attention recently is to utilize multiple channels to improve network performance. In this section, we review and summarize the previous work on multi-channel routing protocols and load balancing techniques that are relevant to our work.

Many routing protocols have been proposed for multi-hop networks, that supports only a single-channel [19]. Recently, routing protocols have been proposed for multi-channel multi-hop networks,



(a) MCP-LB (Proposed), 4 channels



(b) MCP, 4 channels

Figure 8.15: Weighted load at each AP over time.

that combine channel assignment and routing so that multiple channels can be utilized without changing the MAC layer protocol. MCRP, proposed in Chapter 6, is an example of a multi-channel routing protocol for multi-hop networks. Draves et al. [15] proposed a metric for route selection in multi-channel network. The metric, called Weighted Cumulative Expected Transmission Time (WCETT), selects high quality routes considering bandwidth and loss rate of the link, and also the amount of interference on the channel. This protocol assumes that each node has the number of interfaces equal to the number of available channels. Kyasanur et al. [60] proposed a routing protocol that requires multiple network interfaces per node, the number of interfaces does not need to equal the number of available channels. Among multiple interfaces, each node maintains one interface on a fixed channel so that neighboring nodes know on which channel it should transmit to reach this node. The other interfaces are free to switch channels. Raniwala et al. [17, 51] proposed a multi-channel routing protocol that also requires multiple interfaces per node. The authors address two main issues: neighbor-interface binding and interface-channel assignment. Since two neighboring nodes need to be on the same channel to communicate, these nodes need to have at least one interface that is on a common channel. Within this constraint, the protocol tries to assign channels to interfaces so that the load is balanced among channels.

The proposed protocol also assigns channels at the network layer, and is most similar to MCRP (discussed in Chapter 6) and [17]. The protocol is similar to MCRP in the sense that the protocol assumes a single NIC per node. However, MCRP assumes no infrastructure, and supports on-demand route establishment between any two nodes in the network if they need to communicate. Instead, the MCP-LB protocol optimizes for when an infrastructure exists and only the routes between APs and mobile nodes need to be maintained proactively. With the infrastructure, two mobile nodes can communicate if they are connected with an AP. As a result, the MCP-LB protocol does not need nodes that switch channels frequently, which reduces channel switching overhead.

Also, MCP-LB is similar to [17], because it assumes existence of infrastructure, and maintains routes between mobile nodes and access points. Also, the goal of MCP-LB is to balance the load among channels, so that the channel utilization is maximized. There are some differences between [17] and our work. MCP-LB does not require multiple interfaces per node. Supporting nodes with single interface can be beneficial because equipping multiple network interface can be costly for small and cheap devices. Also, we use a different metric for estimating load in the route tree.

Finally, we review the load balancing techniques proposed for wireless networks. Hsiao et al. [61] at el. proposed a load balancing algorithm for wireless access networks. The protocol builds a backbone tree rooted at the APs, similar to our proposed protocol. However, in [61], the AP directs nodes to switch to another tree. This is not possible if the AP does not have the neighbor information of all the nodes, because AP does not know what alternative routes the node can take if it decides to switch trees. In MCP-LB, each node independently decides whether it should switch to another tree. Hassanein et al. [62] proposes to use as the number of "active" paths in the neighborhood as the load metric. Also, Lee et al. [63] use the number of packets buffered in its interfaces as the load metric. We argue in Section 8.1 that locally measured load may not reflect the actual load, and propose the *weighted-load* metric.

8.5 Summary

In this chapter, we looked at utilizing multiple channels in a hybrid network. A hybrid network consists of access points that have wired connection, and mobile hosts that can be connected to the access points via multiple hops. In hybrid networks, mobile hosts only need to maintain routes to access points. We proposed a routing and channel assignment protocol for hybrid network, where traffic is balanced among available channels. The protocol ensures that every node in the network has at least one route to an AP, while allowing nodes to switch channels to associate with an AP with minimum load (the APs are assigned to fixed channels.) We have argued that locally measured load may not reflect actual load, because the node does not know whether the low traffic indicates congestion near the AP, or the requested traffic was low in the first place. Thus, we proposed a load metric that considers number of hops from AP to the destination.

Using the proposed load metric, the load information is distributed via control messages, and each node can independently decide to switch channels so that it can join a route tree with minimum load. As a whole, the network adapts to changing traffic conditions and balances load among channels. The simulation results show that our proposed protocol can successfully reduce congestion in hot-spots and avoid wasting channel bandwidth due to unbalanced traffic load.

Chapter 9

Prototype Implementation

Until now, we have looked at issues in designing a multi-channel multi-hop network, and proposed link-layer and network-layer protocols that can utilize multiple channels with single interface. The performance evaluations for the proposed protocols were done by simulations using ns-2 simulator. Simulations can provide estimates on how the protocols will perform if they are implemented in real systems, but still cannot capture many practical issues that exist in real situations. For example, some features that are easy to implement in simulations may not be easy to implement in real systems. Also, assumptions made in simulations may not hold in real scenarios. Because of this, implementing protocols in a prototype system and experimenting in real scenarios gives much more confidence that the protocols will actually work.

As a proof-of-concept, we have implemented the channel switching features in MCRP, a multi-channel routing protocol presented in Chapter 6. MCRP is an on-demand routing protocol similar to AODV, but designed to utilize multiple channels with single interface. For the prototype implementation, we have used *Net 4521* boxes from Soekris [64] running Linux (kernel version 2.4.26). A Soekris box is equipped with a wireless card based on Atheros chipset [65], which supports IEEE 802.11a/b/g standards. The implementation was done in two parts: a user-space module and the device driver. For the device driver, we have modified the *madwifi* driver [66] to add features needed for multi-channel protocols. In this chapter, we present the details of our implementation and discuss issues we have found during the implementation.

9.1 Background

9.1.1 The Basic Features of MCRP

In basic MCRP, a node may be assigned at most two channels, if multiple flows are overlapping at the node. if a node is assigned two channels, it becomes a *switching node*. Other nodes that are assigned only a single channel are *fixed nodes*. The constraint of MCRP is that two consecutive nodes in a path cannot be switching nodes. Consider the scenario shown in Figure 9.1. Node A is sending traffic to C, and node D is sending traffic to E. Both flows use node B as their relay. In MCRP, these two flows may be assigned a different channel, and node B becomes a switching node. Suppose the flow from A to C is assigned channel 1 and the flow from D to E is assigned channel 2. Then, node B has to switch between channel 1 and channel 2 according to a scheduling algorithm (fixed scheduling in basic MCRP).

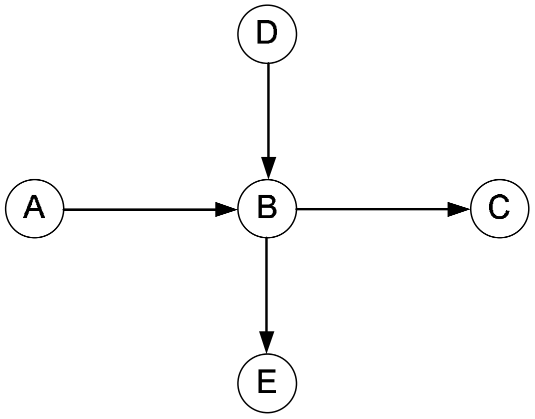


Figure 9.1: An example network scneario with 5 nodes to illustrate features of MCRP.

Whenever node B switches channels, it sends out LEAVE and JOIN messages. When node B switches from channel 1 to channel 2, B broadcasts a LEAVE message on channel 1, switches to channel 2, and broadcasts a JOIN message on channel 2. When node A, a fixed node on channel 1, receives the LEAVE message from B, it defers transmission and buffers the packets for B until it receives a JOIN message. When node A receives a JOIN message, it can resume sending packets to B.

In order to implement this channel switching feature, several issues must be addressed. They are listed in the following.

- Node A must know if node B is currently on channel 1 or another channel. This means that channel state must be kept for each destination. This state is changed whenever a node sends or receives JOIN/LEAVE messages.

- When node B is on channel 1, it may send packets to node C, but it should buffer packets for node E until it switches to channel 2. This means that node B must be capable of selecting the next packet for transmission based on its destination.

- When node B decides to send a LEAVE message on channel 1, it must send the LEAVE message ahead of other packets buffered to be sent on channel 2 later.

- When a node switches channels, it should not flush the packets.

In the next section, we discuss the process of a packet transmission in the Linux operating system.

9.1.2 The Packet Transmission Process in Linux Operating System

When a user application sends a packet using the socket structure, the packet is passed down to the Internet Protocol (IP) implemented in kernel space. At the IP layer, the kernel looks for a valid route for the destination specified in the packet [67]. After a route has been found, the IP header is filled out and the packet is passed down to the link layer. At the link layer, the MAC header is attached to the packet, and the packet is queued at the interface queue for the network device driver to pick up whenever the network card is ready. The network device driver, whenever it is ready, deques a packet from the queue and transmits the packet based on IEEE 802.11 MAC protocol. The packets in the queue are processed with a First-In, First-Out (FIFO) policy. When a packet transmission is finished, the network card raises an interrupt, and the interrupt handler in the device driver transmits the next packet in the queue.

To implement MCRP, we need several modifications to the device driver. First, we need to be able to select which packets to transmit next, based on the current channels of the sender and the receiver. One design choice is to maintain a separate queue for each neighbor. The disadvantage of this approach is that the packets may not be transmitted in the order they are arrived, even when all the receivers are ready to receive the packet. Another approach is to maintain a more sophisticated buffer, where we can extract the first packet that can be transmitted based on current channel status. Second, we need to store a table that has an entry for each neighbor and its current channel status, and also the channel status of the node itself. Since the channel status might change after tha packets have been brought down to the interface queue, the table must be stored somewhere below the queue. One approach is to maintain the channel table in the device driver. When the node decides to switch channels or if it receives a JOIN or LEAVE message, it can update the entry in the channel table using an ioctl call. The second approach is to maintain the channel status at the routing table, by modifying the table to include an additional field for channel status. Then, another table can be maintained at the device driver, which does not store channel information, but only marks whether a neighbor node is ready to receive a packet or not. The second approach needs slightly more memory space, but it is a more general approach because the table at the device driver can be used for other purposes, such as buffering packets for destinations that are currently sleep for the purpose of power saving. Finally, the JOIN and LEAVE messages can be generated in the device driver, or a user-space module can take care of generating and sending these messages. Since JOIN and LEAVE messages are protocol specific and MCRP is a routing protocol, it would be more proper to generate and send these messages at the network layer.

9.2 Implementation Details

In this section, we describe the details of our prototype implementation. Our implementation consists of a user-space module and a modification to the device driver. The user-space module manages the *channel table*, a table that stores the channel information for each neighbor. For switching nodes, it generates and sends JOIN and LEAVE messages whenever it switches channels. Also, it runs a scheduling algorithm and maintains a timer that controls channel switching. Currently, we have implemented a fixed scheduling algorithm, in which the node switches channel after a fixed time interval. For fixed nodes, the module receives and handles the JOIN and LEAVE mssages. The modified device driver has a neighbor table that marks whether a neighbor is currently on the same channel as the current node. Also, the device driver has a buffer where it stores packets, and can extract packets based on the neighbor table.

Before explaining each component of our implementation, we want to note that our device driver implementation is based on the madwifi device driver [66] that can work with network cards based on Atheros chipsets. On top of the basic driver implementation, we have used the modification to channel switching procedure made by Chereddi et al. [68]. The modification was to reduce the channel switching delay. Originally, when a node switches channels, it listens for beacons that advertises a network with the same ESSID as the node itself. If such a beacon is received, the node sets the BSSID to match that of the beacon sender. If no beacon of the same ESSID is received within a certain amount of time, the node creates a new BSSID and start sending beacons. While the delay for channel switching is small, this process of listening and advertising creates a large delay. Since channel switching does not mean switching to a different network in multi-channel environment, we do not need this listening and advertisement procedure. Chereddi et al. removed this beacon features and achieved less than 5 ms of switching delay. We use this modification to build our implementation.

Figure 9.2 presents an overview of our implemenation. There are three subcomponents in the user-space module: channel table, channel timer, and message handler. The entries in the channel table are filled whenever it receives a message from a new neighbor. If the node is a switching node, then the channel timer controls the channel switching time. When the timer expires, the message handler is called to send out a LEAVE message. After sending a LEAVE message, the module makes an ioctl call to the device driver to switch channels. Then, the message handler sends a JOIN message. If a JOIN or LEAVE message is received, the message handler updates the channel table accordingly. Whenever the channel table is updated, the module makes an ioctl call to update the neighbor table in the device driver.

The device driver has two components: neighbor table and packet buffer. The neighbor table is managed by ioctl calls from user-space module, and is looked up every time the driver tries to fetch the next packet for transmission. The packet buffer is used to store packets before transmission, and selectively extract packets that can be transmitted currently. This buffer replaces the interface queue that sits between the IP stack and the device driver, in the Linux kernel. Thus, whenever there is a packet in the interface queue, the device driver deques the packet from the interface queue and store the packet in its own packet buffer. When the network card becomes idle, the

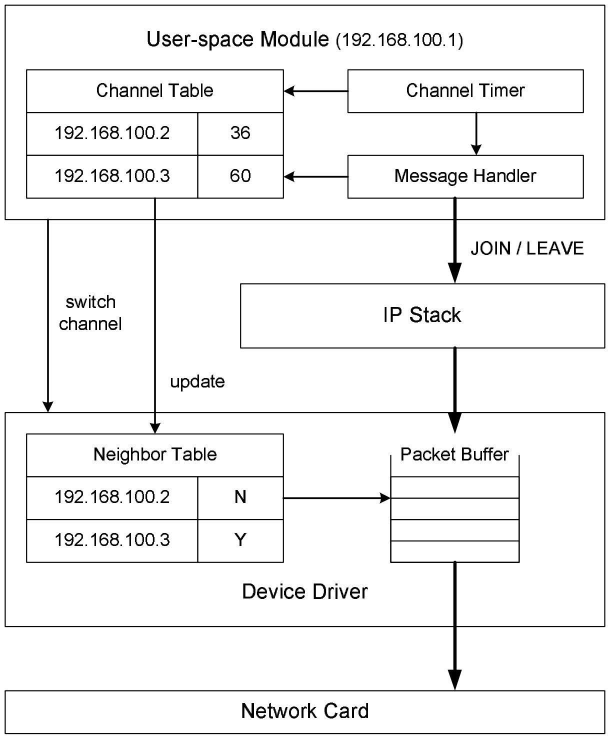


Figure 9.2: The architecture of prototype implementation.

device driver looks up the neighbor table and deques from the packet buffer a packet that can be sent immediately. When the device driver receives an ioctl call from the user-space module asking to switch a channel, the driver waits for the LEAVE packet. When the LEAVE packet is passed down from the user-space module to the driver, it enques the LEAVE packet in the front of the buffer, so that it can be transmitted as soon as the network card becomes idle. Once the LEAVE packet is transmitted, the driver switches channel, and waits for the JOIN packet. After the driver transmits the JOIN packet, it can resume sending packets on the new channel. In our experiment, we set the time for staying on a channel to be 500ms.

In some cases, packets in the queue may stay forever if the receiver has moved away or stopped operating. To avoid packets from taking up buffer space without being transmitted, packets are discarded if they are not transmitted for a certain duration of time (we use 5 seconds). This duration must be much longer than the channel switching interval, so that packets do not get discarded even if the destination is alive.

Since our implementation needs to maintain tables in the user-space module and the device driver, there is memory overhead. The table in the user-space module records the current channel for each neighbor. In our implementation, 4 bytes are used for the IP address, and another 4 bytes are used for the channel number. Thus, for each neighbor, 8 bytes are used. The table in the device driver records a flag for each neighbor, which tells whether the neighbor is ready to receive packets from the current node. The flag is 1 byte, and the IP address is 4 bytes. Thus, for the device driver, 5 bytes are required for each neighbor. In total, 13 bytes are required for each neighbor. If a node has 100 neighbors, then 1.3 Kbytes of memory space is required for our implementation architecture. The amount of required memory depends on the number of neighbors, but even for a dense network, we need only a small memory to implement our architecture.

9.3 Experiment

We have performed an experiment to validate the feasibility of our implementation. The scenario used in our experiment is shown in Figure 9.3. 4 nodes were placed inside a small area so that any pair of node can communicate directly. Each node is equipped with a single IEEE 802.11a card. Among 4 nodes, node A and node C are configured to be fixed nodes, whereas node B and node C are configured to be switching nodes. (In MCRP, nodes automatically decide whether it should be a fixed node or a switching node during route discovery phase. As mentioned earlier, we only implement the channel switching feature of MCRP in our implementation, and do not implement route discovery process.) We use two channels among IEEE 802.11a channels, channel 36 and channel 60. Node A is fixed on channel 36 and node C is fixed on channel 60. Nodes B and D switches between these two channels. The bit-rate for both channels are set to 6Mbps. The channel switching period for the switching nodes is set to 500 ms. (The switching delay is approximately 5 ms.)

Using the Iperf tool [73], each node generates 10Mbps of CBR traffic for one of its neighbors. Node A sends traffic to node B, B sends traffic to C, C sends traffic to D, and D sends traffic to A. When running the experiment, we had nodes B and D start at the same time, with B's initial channel set to 36 and D's initial channel set to 60. This is to have links AB and CD active at the same time on different channels, and to have link BC and AD active at the same time. The experiment was run for 1 minute.

For the purpose of comparison, we also run an experiment for single-channel case, where all 4 nodes are fixed on channel 36. The flow throughput of single-channel case and multi-channel case are compared.

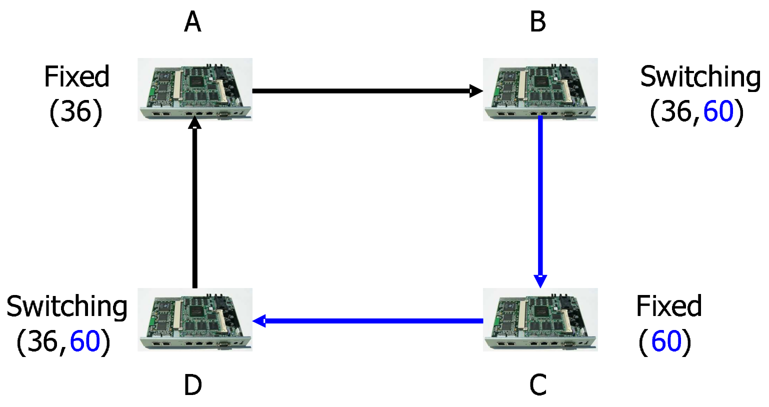
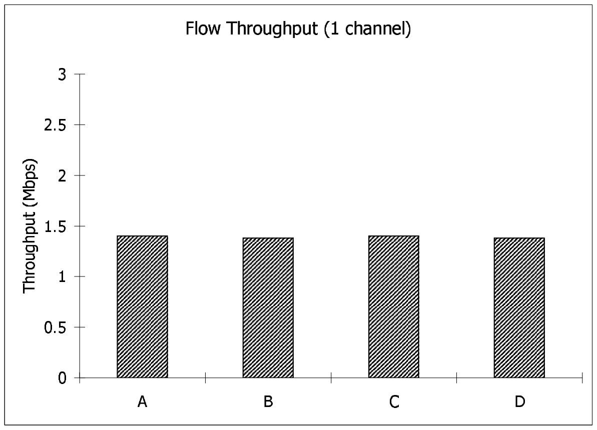
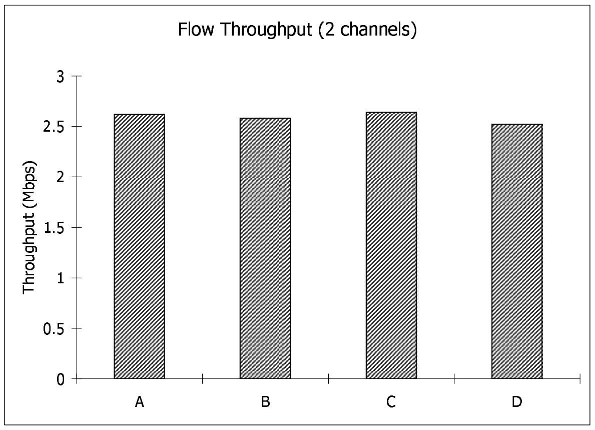


Figure 9.3: Experimental setup: a 4-node scenario. Node A is fixed on channel 36, node C is fixed on channel 60, and node B and D switch between the two channels.

The results are shown in Figure 9.4. Figure 9.4(a) shows the flow throughput when a single channel is used, and Figure 9.4(b) shows the flow throughput when two channels are used. When two channels are used, the throughput for each flow was higher than single-channel case by almost a factor of 2 (slightly lower due to channel switching overhead and control message overhead.) This result shows that it is feasible to utilize multiple channels with single interface using channel switching.



(a) Throughput with a single channel.



(b) Throughput with two channels.

Figure 9.4: Experimental result for 4-node scenario shown in Figure 9.3.

Chapter 10

Conclusion

The multi-channel multi-hop network architecture is a promising candidate for next generation wireless network. In this architecture, it is likely that the number of network interfaces at each node is less than the available number channels. In this thesis, we aim to investigate the case where nodes are equipped with a single interface. MAC and routing protocols that are designed for single-channel networks cannot fully utilize the available spectrum because they rely on the fact that all nodes are listening on a common channel. Thus, we need a different set of protocols to make use of multiple channels and achieve high performance.

Multi-channel protocols must address three major issues: channel synchronization, channel assignment and channel scheduling. When the sender transmits a packet, it must be sure (with high probability) that the receiver is on the same channel as itself. Unsynchronized channels can cause significant performance degradation. Channel assignment should be done in a way that channel diversity is highly exploited. However, to achieve high channel diversity, the protocol needs high overhead of gathering information on channel quality, which results in a trade off. If there is a case when a node needs to send packets to multiple destinations with different channels, the node must schedule time on each channel properly so that packets are fairly delivered. The trade-off here is between the channel switching delay, and the variance in packet delay (burstiness).

In Chapter 3, we have looked at multi-channel MAC protocols that combine channel assignment with medium access control. Based on the discussions, we have developed the MMAC protocol, which allocates channels based on periodic channel negotiation. The MMAC protocol can utilize multiple channels with a single transceiver, but the protocol requires clock synchronization among nodes. Motivated by this requirement, we have developed a light-weight multi-hop clock synchronization service in Chapter 4. The MTSF protocol can exchange MAC-level messages to maintain low clock error between any pair of nodes in the network. In Chapter 5, we integrated MMAC with MTSF and also introduced an extension to AODV that tries to find disjoint routes. AODV-E performs better than AODV when working with MMAC, because MMAC tends to allocate a single channel for all nodes within a flow. In Chapter 6, we looked at multi-channel routing protocols that perform channel assignment and coordination at the network layer. We have developed MCRP, a routing protocol for ad hoc networks that utilize multiple channels with a single transceiver, and can be used with the conventional IEEE 802.11 DCF which does not have any consideration on channels. In Chapter 7, we developed a link-layer channel assignment and scheduling protocol that can emulate two interfaces using a single interface by switching channels synchronously. Compared to MMAC and MCRP, this protocol can assign different channels to nodes within a flow, thereby reducing in-flow self-contention and selecting best channels based on spatially non-uniform channel quality. In Chapter 8, we looked at hybrid networks, where access points exist in a multi-hop network. In hybrid networks, it is enough to establish and maintain at least one route to an access point. So the routing protocol can be made simpler than a protocol for general ad hoc networks. We have developed a routing and channel assignment protocol for hybrid networks, that balances load among different access points so that available channel spectrum is used as much as possible. Finally, in Chapter 9, we implemented the basic features of MCRP in a wireless network testbed based on Soekris Net 4521 boxes running Linux operating system. The prototype implementation provided insights on how to implement multi-channel protocols in real systems.

Many open problems exist in the area of multi-channel networking. The focus of this dissertation is on the scenario where nodes are equipped with a single interface. A vast amount of research is going on for cases where nodes have multiple interfaces. One future research direction is to investigate utilizing multiple channels that have different power, rate and propagation characteristics. There have been efforts to address this problem, but there is still much to be done. Another direction is to apply different protocols for different channels, such as using CSMA/CA for some channels and TDMA for other channels. This approach will be especially interesting in the context of Quality-of-Service, because different channels can be managed to provide different level of quality in terms of delay, jitter and throughput.

References

[1] J. So and N. H. Vaidya, "Multi-channel mac for ad hoc networks: Handling multi-channel hidden terminals using a single transceiver," in *ACM Mobihoc*, 2004.

[2] J. So and N. H. Vaidya, "MTSF: A light-weight timing synchronization protocol to support synchronous operations in multi-hop wireless networks," tech. rep., University of Illinois at Urbana-Champaign. December 2004.

[3] J. So and N. 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.

[4] J. So and N. Vaidya, "Utilizing multiple channels with a single interface: A link-layer channel assignment and scheduling protocol for multi-channel multi-hop wireless networks," tech. rep., University of Illinois at Urbana-Champaign. January 2006.

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

[6] Mobile Ad-hoc Networks (MANET) Charter. "http://www.ietf.org/html.charters/manet-charter.html,".

[7] Mesh Networks Inc. "http://www.meshnetworks.com,".

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

[9] Bay Area Wireless Users Group. "http://www.bawug.org,".

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

[11] Y. Lin and Y. Hsu, "Multihop cellular: A new architecture for wireless communications," in *IEEE INFOCOM*, 2000.

[12] W. List and N. Vaidya, "A routing protocol for k-hop networks," in *WCNC*, 2004.

[13] Z. L. B. Liu and D. Towsley, "On the capacity of hybrid wireless networks," in *IEEE INFOCOM*, 2003.

[14] C. Qiao and H. Wu, "icar: An intelligent cellular and ad-hoc relay system," in *IEEE IC3N*, 2000.

[15] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *ACM Mobicom*, 2004.

[16] P. Kyasanur and N. Vaidya, "Routing and interface assignment in multi-channel multi-interface wireless networks," in *IEEE WCNC*, 2004.

[17] A. Raniwala and Tzi-cker Chiueh", "Architecture and algorithms for an ieee 802.11-based multi-channel wireless mesh network," in *IEEE INFOCOM*, 2005.

[18] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, 1999.

[19] E. Royer and C. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," *IEEE Personal Communications*, April 1999.

[20] C. Perkins and P. Bhargwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *ACM SIGCOMM*, 1994.

[21] T. Clausen and P. Jacquet, "Optimized link state routing protocol," *Ietf Manet Working Group (Draft 11)*, 2003.

[22] D. B. Johnson, D. A. Maltz, and Y.-C. Hu, "The dynamic source routing protocol for mobile ad hoc networks (DSR)," *IETF Manet Working Group (Draft 10)*, 2004.

[23] Z. Haas, M. Pearlman, and P. Samar, "The zone routing protocol (ZRP) for ad hoc networks," *Ietf Manet Working Group (Draft 4)*, 2002.

[24] J. Deng and Z. Haas, "Dual busy tone multiple access (DBTMA): A new medium access control for packet radio networks," in *IEEE ICUPC*, 1998.

[25] Z. Tang and J. J. Garcia-Luna-Aceves, "Hop-Reservation Multiple Access (HRMA) for Ad-Hoc Networks," in *Proc. of IEEE INFOCOM*, 1999.

[26] A. Tzamaloukas and J.J. Garcia-Luna-Aceves, "A Receiver-Initiated Collision-Avoidance Protocol for Multi-Channel Networks," in *Proc. of IEEE INFOCOM*, 2001.

[27] A. Nasipuri, J. Zhuang, and S. Das, "A multichannel csma mac protocol for multihop wireless networks," in *WCNC*, September 1999.

[28] A. Nasipuri and S. Das, "Multichannel csma with signal power-based channel selection for multihop wireless networks," in *VTC*, September 2000.

[29] S.-L. Wu, C.-Y. Lin, Y.-C. Tseng and J.-P. Sheu, "A New Multi-Channel MAC Protocol with On-Demand Channel Assignment for Multi-Hop Mobile Ad Hoc Networks," in *Int'l Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN)*, 2000.

[30] 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.

[31] I.A. Getting, “The Global Positioning System,” *IEEE Spectrum 30*, December 1993.

[32] W. Hung, K. Law and A. Leon-Garcia, “A Dynamic Multi-Channel MAC for Ad Hoc LAN,” in *Proc. of 21st Biennial Symposium on Communications*, April 2002.

[33] VINT Group. “UCB/LBNL/VINT network simulator ns (version 2),”.

[34] The CMU Monarch Project. “Wireless and Mobility Extension to ns,”.

[35] H. Woesner, J. Ebert, M. Schlager and A. Wolisz, “Power-saving mechanisms in Emerging Standards for Wireless LANs: The MAC Level Perspective,” *IEEE Personal Communications*, June 1998.

[36] E.-S. Jung and N. H. Vaidya, “An Energy Efficient MAC Protocol for Wireless LANs,” in *Proc. of IEEE INFOCOM*, June 2002.

[37] A. Nasipuri, S. Ye, J. You and R. Hiromoto, “A MAC Protocol for Mobile Ad Hoc Networks using Directional Antennas,” in *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, Chicago, IL, September 2000.

[38] L. Huang and T. Lai, “On the Scalability of IEEE 802.11 Ad Hoc Networks,” in *ACM MOBIHOC*, June 2002.

[39] Y.C. Tseng, C.S. Hsu and T.Y. Hsieh, “Power-Saving Protocols for IEEE 802.11-Based Multi-Hop Ad Hoc Networks,” in *IEEE INFOCOM*, 2002.

[40] C. Schurgers, V. Tsiatsis, S. Ganeriwal and M. Srivastava, “Topology Management for Sensor Networks: Exploiting Latency and Density,” in *ACM MOBIHOC*, June 2002.

[41] D. Mills, “Internet Time Synchronization: The Network Time Protocol,” *IEEE Trans. Communications.*, October 1991.

[42] J. Elson and D. Estrin, “Time Synchronization for Wireless Sensor Networks,” in *International Parallel and Distributed Processing Symposium (IPDPS)*, April 2001.

[43] J. Elson, L. Girod, and D. Estrin, “Fine-Grained Network Time Synchronization using Reference Broadcasts,” in *Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[44] Kay Römer, “Time Synchronization in Ad Hoc Networks,” in *ACM MOBIHOC*, October 2001.

[45] M.L. Sichitiu and C. Veerarittiphan, “Simple, Accurate Time Synchronization for Wireless Sensor Networks,” in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2003.

[46] Jana van Greunen and Jan Rabaey, “Lightweight Time Synchronization for Sensor Networks,” in *Workshop on Wireless Sensor Networks and Applications (WSNA)*, September 2003.

[47] S. Ganeriwal, R. Kumar, M. B. Srivastava, “Timing-sync Protocol for Sensor Networks,” in *First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[48] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," in *IEEE INFOCOM*, 2004.

[49] J.-P. Sheu, C.-M. and C.-W. Sun, "A Clock Synchronization Algorithm for Multi-Hop Wireless Ad Hoc Networks," in *IEEE ICDCS*, 2004.

[50] R. Fan, I. Chakraborty, and N. Lynch, "Clock synchronization for wireless networks," in *Proceedings of 8th International Conference on Principles of Distributed Systems*, December 2004.

[51] A. Raniwala, K. Gopalan, and T. Chiueh, "Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks," *Mobile Computing and Communications Review*, vol. 8, pp. 50–65, April 2004.

[52] N. Shacham and P. King., "Architectures and performance of multichannel multihop packet radio networks," *IEEE Journal on Selected Area in Communications*, vol. 5, pp. 1013– 1025, July 1987.

[53] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou, "A multi-radio unification protocol for ieee 802.11 wireless networks," in *IEEE International Conference on Broadband Networks (Broadnets)*, 2004.

[54] R. Chandra, P. Bahl, and P. Bahl, "Multinet: Connecting to multiple ieee 802.11 networks using a single wireless card," in *IEEE Infocom*, Hong Kong, March 2004.

[55] R. Choudhury and N. Vaidya, "Deafness: A mac problem in ad hoc networks when using directional antennas," in *IEEE ICNP*, 10 2004.

[56] Z. Ye, D. Berger, P. Sinha, S. Krishnamurthy, M. Faloutsos, and S. K. Tripathi, "Poster abstract: Alleviating mac layer self-contention in ad-hoc networks," in *Mobicom*, 2003.

[57] 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.

[58] Saurabh Ganeriwal, Ram Kumar, Mani B. Srivastava, "Timing-sync Protocol for Sensor Networks," in *ACM Sensys*, November 2003.

[59] C. Tschudin, R. Gold, O. Rensfelt, and O. Wibling, "Lunar - a lightweight underlay network ad-hoc routing protocol and implementation," in *Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN)*, 2004.

[60] P. Kyasanur and N. Vaidya, "Routing and interface assignment in multi-channel multi-interface wireless networks," in *IEEE WCNC*, 2005.

[61] P. Hsiao, A. Hwang, H. Kung, and D. Vlah, "Load-balancing routing for wireless access networks," in *IEEE INFOCOM*, 2001.

[62] H. Hassanein and A. Zhou, "Routing with load balancing in wireless ad hoc networks," in *ACM Int'l workshop on modeling, analysis and simulation of wireless and mobile systems (MSWim)*, 2001.

[63] S. Lee and M. Gerla, "Dynamic load-aware routing in ad hoc networks," in *ICC*, 2001.

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

[65] Atheros inc. "http://www.atheros.com,".

[66] Multiband Atheros Driver for WiFi (MADWiFi). "http://sourceforge.net/projects/madwifi,".

[67] K. Wehrle, F. Pahlke, H. Ritter, D. Muller, and M. Bechler. "The linux networking architecture, prentice hall, 2005,".

[68] C. Chereddi, P. Kyasanur, and N. Vaidya, "Design and implementation of a multi-channel multi-interface network," in *REALMAN*, May 2006.

[69] J. Broch, D. Maltz, and D. Johnson, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *MOBICOM*, 1998.

[70] S. Gwalani, E. Belding-Royer, and C. Perkins, "AODV-PA: AODV with path accumulation," in *Next Generation Internet Symposium*, May 2003.

[71] M. Caccamo, L. Zhang, L. Sha, and G. Buttazzo, "An implicit prioritized access protocol for wireless sensor networks," in *IEEE Real-Time Systems Symposium*, December 2002.

[72] X. Yang, and N. Vaidya, "On the physical carrier sense in wireless ad-hoc networks," in *IEEE INFOCOM*, 2005.

[73] Iperf. "http://dast.nlanr.net/Projects/Iperf,".

[74] S. Biaz, and J. Welch, "Closed form bounds for clock synchronization under simple uncertainty assumptions." in Information Processing Letters, 2001.

Author's Biography

Jungmin So received the B.S. degree in Computer Engineering from Seoul National University, Korea, in 2001. In August 2001, he joined the Computer Science department at University of Illinois at Urbana-Champaign to pursue a Ph.D. in the area of systems and networking. Since then, he has been working on wireless network protocols in the Wireless Networking Group, led by Prof. Nitin Vaidya. His major work during the Ph.D. program was developing protocols for multi-channel multi-hop wireless networks. He was awarded the Graduate Scholarship from Korea Foundation for Advanced Studies from 2001 to 2006, and the Illiac Fellowship from University of Illinois in 2001.