Design of A Multi-Channel MAC Protocol for Ad Hoc Wireless Networks

Hoi-Sheung Wilson So
Jean Walrand

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-17
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-17.html

February 17, 2006

Acknowledgement

Part of this report is based on a joint project with Jeonghoon Mo and published in ACM/IEEE MSWiM 2005. This work was supported in part by the National Science Foundation under Grant CNS-0435478.

Design of a Multiple Channel Medium Access Protocol for Ad-Hoc Wireless Networks

Hoi-Sheung Wilson So

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
so@cs.berkeley.edu

February 17, 2006

Abstract

Typically, a device in an ad hoc wireless network has a tunable radio allowing it to listen or transmit on one channel at a time. To ensure maximum connectivity, all devices tune their radio to the same channel. However, as node density increases, the interference worsens, resulting in lower throughput per device.

Some of the interference is unnecessary. Modern radios can often switch channels (i.e., tune to different frequencies). Given multiple available channels, devices in a neighborhood can potentially transmit on different channels simultaneously to increase the network throughput.

This dissertation explores whether it is possible to exploit the existence of multiple channels and the ability of radios to switch between them quickly to increase network throughput without a central coordinator.

The thesis starts by showing that if a radio can modulate the entire available spectrum, splitting it into smaller channels does not increase capacity. However, since there is always an engineering limit as to how wide a channel can be, multi-channel MAC should be viewed as an orthogonal and complementary way to increase throughput when more spectrum is available, without hardware changes.

Existing multi-channel MAC protocols were classified into four categories based on their operation principle: Dedicated Control Channel, Split-phase, Common Hopping, and Parallel Rendezvous. These four approaches were compared through both analysis and simulation. All approaches apart from Parallel Rendezvous suffer from control channel congestion when channels are numerous and communication tend to be short.

Using the best performing approach, Parallel Rendezvous, as a skeleton, we designed

our own multi-channel MAC protocol called McMAC. To prove that McMAC is practical and to discover any hidden issue, we implemented the salient features of McMAC on an experimental hardware platform.

In summary, through analysis, simulation, and experiments, we show that it is possible to increase throughput of an ad hoc wireless network by using multiple channels efficiently even though each node can only use one channel at a time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Typically, in a wireless data network, all devices use the same channel (i.e. the same frequency) so that they can easily communicate with each other. For example, in a home wireless network, the access point connected to the Internet communicates with all wireless-enabled computers using the same frequency. This approach allows devices to discover each other easily. However, as the number of devices increases, the throughput share of each device decreases due to contention of the wireless medium. However, not all such contention is necessary. Imagine a wireless home network scenario in which a user is streaming a video stored on a DVD player to a television in another room. At the same time, others are playing a video game running on a PC by sending the screen output to a projector wirelessly. Both the video stream from the DVD player to the TV and that from the PC to the projector have to compete for the wireless medium. In this scenario, if two channels are available, the two pairs of devices can easily use two different channels so each stream enjoys the full speed of the radio. However, today's technology only give users two choices. Either all 4 devices use the same channel and suffer low throughput, or each pair uses a different channel but then the DVD player can never communicate with the projector. The first choice may not seem so bad until the number of devices increases, which reduces the share of throughput each device gets proportionally. When devices and channels are numerous, the gain in the total network throughput by flexibly using multiple channels together can be tremendous.

This dissertation explores whether it is possible to exploit the existence of multiple

channels and the ability of radios to switch between them quickly to increase network throughput without a central coordinator.

Our research shows that it is possible to design a distributed medium access control (MAC) protocol which coordinates nodes to use multiple channels efficiently. Using this protocol, each node hops across all channels in a pseudo random fashion independently of each other. This enables the load balancing of data and control traffic on all available channels. This protocol does not require any change to existing commercial off-the-shelf radios. Only software modifications are necessary.

Multi-hop wireless networks and mobile ad hoc networks present many additional challenges such as routing and the hidden terminal problem. In this thesis, we only explore the problem in a 1-hop neighborhood. The results of our research can be the building block of a multi-hop version of the multi-channel MAC protocol. Care has been taken to avoid making assumptions that would invalidate our approach upon such extension.

1.2 Motivation

1.2.1 Motivating Scenario

Wireless communication is a very valuable tool in disaster relief scenarios. After a disaster strikes, existing wired infrastructure and even cellular phone networks are often destroyed due to power generator failures and cable cuts. To effectively coordinate different agencies in a disaster relief scenario, it is very important to quickly restore communications to the affect area such that damages can be assessed and resources can be directed to people needing the most help.

In this situation, wireless ad hoc networks is a promising technology. In contrast to wired networks, ad hoc networks can self-organize the wireless relay nodes within the network to find quickly a path from any source to any destination without human supervision. A multi-hop ad hoc network forwards data packet in a bucket brigade fashion along the chosen path.

Inefficiencies of Single Channel MACs

Today, commodity wireless local area network equipment operate at speeds ranging from 11Mbps to 54Mbps per channel. In a multi-hop wireless network, the throughput achievable

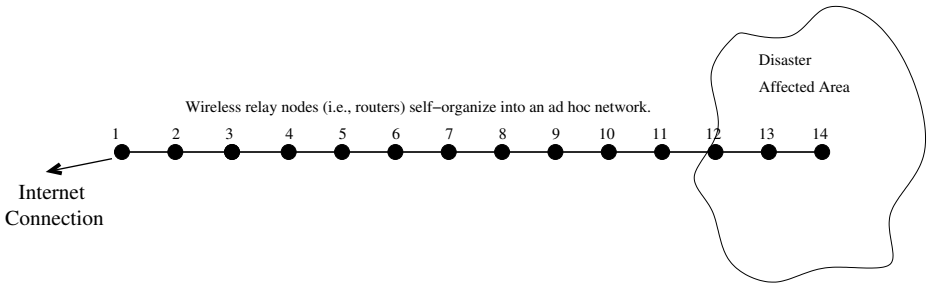


Figure 1.1. Temporary relay routers with the left most node having a direct high speed connection to the Internet.

is much lower than the speed of the radio because the indicated speed does not take into account the medium sharing effects of a multi-hop wireless network. First, nodes cannot transmit and receive at the same time. Second, when a node transmits, all of its neighbors cannot receive from another node. Fortunately, quite often, several channels are available. One can increase network throughput by allowing adjacent nodes to transmit on different channels simultaneously. Consider the disaster relief scenario mentioned earlier. To setup a temporary wireless network to carry information to an area that has been cut off from the Internet or phone network, a number of wireless relay devices are placed linearly along a road to form a chain network as shown in Fig.1.1. Each node in the network has one radio tunable to one of the two available channels. The leftmost device has a direct high speed wired connection to the Internet. Consider the case in which the rightmost node of the network needs to download a large document via the Internet connection. For simplicity, we model it as a unidirectional flow of packets being forwarded from the leftmost node to the right. The goal to maximize the throughput of the flow. Due to interference, for a node to receive a packet from its left neighbor correctly, it cannot be within 2 hops of another sender.

For simplicity, assume that time is slotted, all the packets are of equal length, and the nodes are synchronized. In this situation, if only 1 channel can be used, nodes 1, 5, 9 ... can transmit at the same time while the other nodes must remain quiet (see Fig.1.2). In the next time slot, nodes 2, 6, 10 ... can transmit. During each time slot, the fraction of nodes transmitting is 1/4, and hence the throughput should be approximately 1/4 of the link speed. A similar result was verified in a testbed in [15]. Next, consider the case where 2 channels (twice the total bandwidth) are used. Nodes 1, 5, 9 .... can transmit on channel 1 while nodes 3, 7, 11 ... transmit on channel 2. In the next time slot, nodes 2, 6, 10 ... transmit on channel 1 while nodes 4, 8, 12, ... send on channel 2 (see Fig.1.3). The throughput of the network has doubled from 1/4 to 1/2 of the link speed. Clearly,

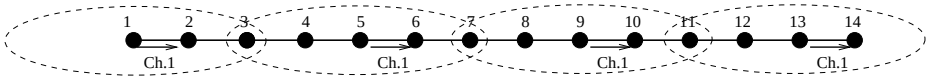


Figure 1.2. Only 1 channel is available.

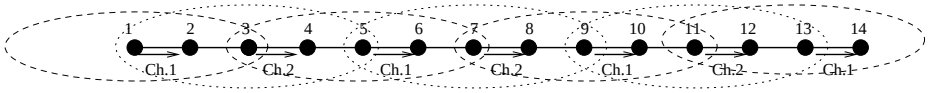


Figure 1.3. Two channels are available.

a distributed multi-channel MAC protocol using 2 channels is unlikely to achieve exactly twice the throughput of a single channel MAC in a network of arbitrary topology. However, the example shows there is a huge potential for network throughput to increase significantly when multiple channels are available. Using today's technology, however, only 1 channel is used even though many channels are usually available. Note that in the preceding example, we have doubled the throughput without increasing the transmission speed of each radio.

For a network using M channels, the network throughput can potentially be increased by M times under ideal conditions. In reality, the gain will be less than M times, but if M is large, the network throughput is still significantly higher than using a single channel only. For example, in the U.S., IEEE 802.11a standard [10] defines 12 non-overlapping channels each of 54Mbps. Even though in practice, usll all 12 channels will not achieve 12 times the throughout (54×12=648Mbps), the potential increase is still very large.

1.2.2 Single Wideband Channel vs. Multiple Channels

A Medium Access Control (MAC) protocol is a protocol which coordinates the access to the transmission medium among different senders. For a wireless network, the medium is usually defined by a range of frequency also known as a band or a channel that a device can use to transmit information. A natural question is why one divides up the available bandwidth into several channels, and then invents a MAC protocol that uses multiple channels. Given a fixed amount of available spectrum, one can envision a wide band radio transmitting information over the entire spectrum forming a single *super-channel*. When only one channel is available, a traditional single channel MAC protocol suffices. The obvious question is whether there is any gain in breaking down a wide band channel into several narrower bands.

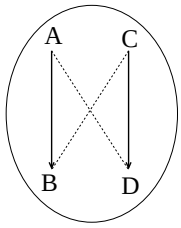


Figure 1.4. A simple 4-node network.

Simple Case

Consider first a very simple case shown in Fig.1.4 with 4 nodes. Node A sends to B only while C sends to node D exclusively. Nodes A and C are very close to each other while B and D are close to each other. Assume that each device transmits at a maximum power spectral density of P[W/Hz] in an environment with background white noise having a constant spectral density of N_o[W/Hz]. Due to path loss, the average received power is P_r[W] where $P_r < P$. We further assume that a fixed bandwidth of W[Hz] is available for this network. One can create one channel with W[Hz] that is time shared between the two pairs of nodes or two independent channels of $W/2$[Hz] each. (Note: in practice, due to imperfect filters, a guard band is needed between the two channels, but we ignore the issue here.) Information theory tells us that using 1 channel of WHz wide, the capacity of the network is: $C_1 = W \log_2(1 + \frac{P_r}{N_o W})$ which is time shared by 2 node pairs. Using 2 narrower channels each of W/2[Hz], two senders can each have a capacity of $C_2 = \frac{W}{2} \log_2(1 + \frac{P_r/2}{N_o(W/2)})$ which is exactly half of C_1. Therefore, the per sender share of the capacity remains the same.

Network Case

To see that splitting a channel into smaller sub-channels does not increase capacity in general, we consider the following wireless network model. Time is slotted. The network is modeled as a set of device nodes $\mathbf{N}$ and a set of links $\mathbf{L}$ represented as a Boolean connectivity matrix. $\mathbf{L}(u, v)$ is 1 if and only if u can send data to v, where u and v are distinct nodes in $\mathbf{N}$. The interference between links is modeled as a set of conflicts $\mathbf{C}$ where $((s_1, r_1), (s_2, r_2)) \in \mathbf{C}$ iff s_1 cannot send to r_1 when s_2 is sending to r_2. This model of conflict between links is quite general.

We define a schedule $\mathbf{S}(t, f, u, v)$ to be a 4-D boolean matrix specifying who should send

data to whom during each time slot on each channel. If $\mathbf{S}(t, f, u, v)$ equals 1, it means that during time slot t, node u sends to node v using channel f. f is between 1 and M, where M is the number of channels available. M is 1 if the spectrum is not sub-divided into multiple channels. For a schedule to be feasible, two conditions must be satisfied.

Condition 1 states that two interfering links cannot be active at the same time:

$$\forall t \quad \forall f, \quad \sum_{((s_1,r_1),(s_2,r_2))\in\mathbf{C}} \mathbf{S}(t, f, s_1, r_1) + \mathbf{S}(t, f, s_2, r_2) \leq 1$$

Moreover, the radio of each node u can either listen or transmit, but not both, on at most 1 channel during each time slot. Condition 2 states that:

$$\forall t \quad \forall u, \quad \sum_{1\leq f\leq M} \sum_{s} \mathbf{S}(t, f, s, u) + \sum_{1\leq f\leq M} \sum_{r} \mathbf{S}(t, f, u, r) \leq 1$$

To establish that using multiple channels does not provide any gain compared to a single super channel, we prove that a schedule $\mathbf{S}$ that is feasible under an M-channel model is also feasible under a 1-channel model using M times more time slots in a transformed schedule $\mathbf{S}'$. As illustrated earlier, the capacity of a link is double that of a link with half the bandwidth if the transmit power scales with the bandwidth. Therefore, in a 1-channel model, since each channel is M times wider than in the M-channel model, the capacity of each link is also M times higher. It takes only $\frac{1}{M}$ of time to transmit the same amount of information using a super channel. As a result, even though $\mathbf{S}'$ uses M times more time slots as $\mathbf{S}$, both schedules finish in the same amount of real time.

Theorem: the capacity of a network in which the available spectrum is not sub-divided is at least that of a network in which the spectrum is divided into M channels and each node can only use 1 channel at a time.

Proof: Suppose $\mathbf{S}$ is a feasible schedule under an M-channel model. We transform $\mathbf{S}$ into a new schedule $\mathbf{S}'$ which uses only 1 channel but M times as many time slots. For each time slot in $\mathbf{S}$, we schedule the transmission on each of the M channels in M time slots, namely, for $t \geq 1$, $\mathbf{S}'((t-1)M + f, 1, u, v) = \mathbf{S}(t, f, u, v)$.

Since condition 1 is satisfied under $\mathbf{S}$, the set of links active during each time slot on each channel is guaranteed to be free of conflict. As a result, the set of links active during each time slot of $\mathbf{S}'$ is also free of conflict on channel 1. Since there is only 1 channel in $\mathbf{S}'$, condition 1 is satisfied in $\mathbf{S}'$.

$\mathbf{S}$ satisfies condition 2 which implies that a node cannot both transmit and receive on

the same channel during any time slot. As a result, under $\mathbf{S}'$, no node will transmit and receive during the same time slot. Condition 2 is therefore also satisfied under $\mathbf{S}'$.

$\mathbf{S}'$ uses M times as many time slots as $\mathbf{S}$, but since it takes only $\frac{1}{M}$ of time to transmit the same amount of information using a super channel of M times the bandwidth, both schedules finish in the same amount of real time.□

The converse of the theorem is not true because a network consisting of 2 nodes can only achieve $\frac{1}{M}$ of the throughput under the M-channel model compared to a 1-channel model.

1.2.3 Practical Reasons for Multiple Channels

Even though using multiple channels does not provide any capacity gain, there are still three practical reasons for dividing available bandwidth into sub-channels. The first two are administrative while the third is technical. First, multiple wireless networks using the same MAC protocol technology but owned by differen people may co-exist in the same area. This is often the case for home wireless networks in densely populated areas. Using different channels for separate networks allows the networks to coexist without interference. If only one channel is available, nodes from different networks must coordinate with each other so that they do not transmit simultaneously or else the network performance can degrade substantially. Coordination among nodes from separate networks is very difficult when the interfering links across the networks are weak. Current wireless MAC protocols tackle this problem by simply dividing the spectrum into several independent channels so that separate but collocated networks can operate without interference.

The second reason for designing multi-channel MAC is that the spectrum available for a particular wireless network often varies from one country to another. Due to historical and political reasons, it is often impossible to assign the same frequency band in every country. Consequently, a wireless network technology is designed to operate at one of several possible frequency bands depending on location. Furthermore, the bandwidth available varies from one country to another. Instead of defining a different standard for every region, multiple channels are defined in the same standard. A subset of these channels can be used in each region. Usually, each resulting channel is narrower than the entire spectrum available in each country. By standardizing on the bandwidth of each channel, a the same tunable radio silicon chip can be used in many different countries. The set of channels that can be used in various countries is configured in software when the user installs the driver.

The third reason is that there is ultimately a limit as to how much bandwidth a radio can modulate efficiently and economically. The challenge in building a very wide band radio lies in the design of a wide band low-noise amplifier on the transmitter side and a good linear filter on the receiver side. As technology progresses, the bandwidth limit will increase. However, given the widest band radio one can build economically, it is always possible to apply a multi-channel MAC protocol to allow a network to utilize even more bandwidth than is possible with any single node. Therefore, we see multi-channel MAC as both orthogonal and complementary to the use of wide band radios.

In the near future, regulations, engineering limits, and the need for simple spectrum management will continue to exist. We therefore expect new wireless network designs to have multiple channels and make use of tunable radios also. The combination of frequency agile radios and multiple channels provides a fertile ground for performance improvement and motivates our search for an efficient multi-channel MAC protocol that can adapt to the number of available channels and maximize the network throughput.

1.3 Problem Definition

We define a *channel* to be a frequency range over which two nodes communicate. The available bandwidth is divided into M channels of equal bandwidth. Two pairs of nodes communicating on different channels are assumed to not interfere with each other. Each node has one tunable radio which allows it to send or receive on any of the channels. Switching channel takes a fixed amount of time. The radio is half-duplex which means that a node cannot transmit and receive at the same time.

A Medium Access Control (MAC) protocol is a set of rules that nodes follow when accessing the medium. The design of a MAC protocol balances the need for high throughput and low medium access delay with fairness among different nodes. Traditionally, all nodes that communicate with each other use the same channel. A multi-channel MAC protocol is different in that nodes communicate over several channels simultaneously.

Given that multiple channels are available, we want to design a multi-channel MAC protocol that efficiently uses as many channels as possible without assuming a particular traffic pattern.

We limit the scope of the problem by making the following assumptions:

1. **Single Hop** We only consider a single hop wireless network in which all nodes are within the communication range of each other. All of our simulation and analysis are based on this model. However, our protocol specification takes into account hidden nodes and does not break down when applied to a multi-hop network, although the performance may be affected.

2. **No Relay** For a multi-hop network to be useful, packets must be forwarded over multiple hops from the source to the destinations. We do not consider routing at the network layer or forwarding in the link layer. To achieve the best performance, relay nodes should coordinate among themselves the receiving and sending of packets. Our work focuses only on communication between direct neighbors.

3. **Equal Channel Quality** We assume that all channels are equal. This may not be true for networks in which fading is frequency selective. When fading is frequency selective, certain channels work better for some pairs but not others.

4. **Quality of Service** We currently do not consider applications that have strict delay or throughput requirements.

5. **Unicast Only** We focus only on delivering unicast traffic in this thesis. Broadcast is also a very useful communication primitive. For example, the Address Resolution Protocol (ARP) relies on broadcast to resolve the IP address of a node into its MAC address. In this particular case, the neighbor discovery mechanism of a multi-channel MAC protocol can easily provide this function. Fortunately, there is no need for broadcast to support the functionality provided by ARP. However, broadcast is still needed in other cases.

1.4 Organization of Dissertation

The rest of the dissertation is organized into 6 chapters. We first survey the state-of-the-art in various related research areas in Ch.2. We will also provide the background knowledge useful in understanding the chapters to follow. Then, in Ch.3, we describe our plan to tackle the design problem and the rationale behind our approach. Before we design a new protocol, we need to understand the merits and shortcomings of any existing solutions. Therefore, in Ch.4, we classify known approaches to the problem into 4 categories and discuss the pros and cons of each. The evaluation is based on both analysis and simulation.

Next, having understood the advantages of each approach, we choose the most promising approach and design a complete multi-channel MAC protocol in Ch.5. The design rationale will be discussed as well. Next, we present an implementation of a selected subset of the full protocol on an experimental hardware platform in Ch.6. The practical challenges together with our solution to them will be discussed in details. The performance of the protocol will be measured in a series of benchmarks to validate the design. Finally, Ch.7 summarizes the findings of our research and point to open problems that have yet been addressed.

Chapter 2

Background and Related Research

2.1 Multi-Channel MAC Protocols

Multi-channel MAC protocols can be divided into those using a single or multiple transceivers (radios) per node. Another way of categorization, as suggested in [19], is by the mechanism sender-receiver pairs use to agree on a data transfer channel. In such way, multi-channel MAC protocols can be classified into 4 categories:

- **Dedicated Control Channel** Examples are DCA (Dynamic Channel Allocation) [32], DCA-PC (Dynamic Channel Allocation with Power Control) [33] and DPC (Dynamic Private Channel) [9].
- **Split-Phase** Both MMAC [25] and MAP (Multichannel Access Protocol) [6] are examples.
- **Common Hopping** Examples include CHMA (channel hoping multiple access) [31] , CHAT (Channel Hopping multiple Access with packet Trains) [30], and Hop-Reservation Multiple Access(HRMA) [28].
- **Parallel Rendezvous** Examples of this approach include SSCH (Slotted Seeded Channel Hopping) [5] and McMAC [23].

Within the Parallel Rendezvous category, there are two protocols proposed in the research literature – SSCH and McMAC. In both protocols, each node has 1 radio and uses a channel hopping sequence. Although they differ in the way the hopping sequences are used, both

protocols require synchronization. SSCH requires all nodes synchronize their hopping times, while McMAC does not. This paper presents a protocol that is a simplified variant of McMAC.

2.2 Time Sync

Time synchronization is a long-standing problem in distributed systems. In some contexts such as in NTP[18], it refers to the distribution of a reference real-time clock to multiple devices connected by a network. In others, it refers to the establishment of a common clock without any real-time requirements. An example of such protocols is the Flooding Time Synchronization Protocol [16]. In this paper, we are only concerned with pair-wise synchronization between 1-hop neighbors. Therefore, we use the term *synchronization* to refer to the prediction of the current clock of 1-hop neighbors only.

Time synchronization among only 1-hop neighbors is a considerably simpler problem due to the following reasons:

- There is no need to agree on a common clock. Hence, there is no convergence issue as each node estimates their neighbors' clocks separately.
- Queueing and medium access delays happen at only the sender, which is known to the sender.
- The propagation delay is bounded by the longest distance between two possible neighbors.

To account for queueing and medium access delays accurately, the outgoing packets should be time-stamped as closely as possible to the physical layer of the network stack. In this paper, we reuse the time-stamping techniques and implementations by Maroti et al. [16].

Chapter 3

Methodology

3.1 Overview

We approach the MAC protocol design problem in 3 steps: (i) understanding the problem and existing solutions, (ii) designing an improved solution, and (iii) implementing and evaluating our proposed solution.

There has been a large number of existing protocols proposed to date. It is therefore necessary to understand the fundamental differences of various approaches to solving the same problem. Having understood the pros and cons of them, we choose one which works best in most scenarios, and design a complete MAC protocol following that approach. Finally, we attend to the practical problems of implementing the proposed protocol. Through the exercise of implementing a simplified protocol including only the salient features of the full proposal, we hope to discover potential difficulties that were either neglected or intentionally ignored in the analysis or simulation.

3.2 Understanding Existing Approaches

Researchers have proposed a number of protocols that exploit frequency agile radios and multiple available channels in many ad hoc wireless networks to increase capacity. The core design issues of multi-channel MAC protocols are about determining (i) the current channel on which a receiver is listening and (ii) whether the medium on that channel is idle. Different

protocols proposed in the literature differ in how they tackle these two challenges. These choices have a direct impact on the delay and throughput characteristics of the protocol.

Unfortunately, most of these proposals have been compared to the baseline case of a single channel MAC protocol. There have been only scant studies comparing these protocols against each other under identical operating conditions. As a result, it is not clear which design approach would be the most fruitful under what kind of traffic load and network conditions.

We believe that before rushing to propose yet another multi-channel MAC protocol, one should understand how these existing protocols differ from each other fundamentally (if they do), and what performance tradeoffs they imply. Towards this goal, a detailed simulation comparison of each and every protocol would give us the absolute numbers in a handful of scenarios, but not a better overall understanding of how these approaches perform differently under what conditions. An analytical approach to the comparison would give us more insight into the performance of the different protocols, but we inevitably have to make many simplifying assumptions in order to keep the analysis tractable.

We decided to take a hybrid approach. We first generalize the larger number of protocols and classify them into a small number of design approaches. We then build analytical models for each of the few approaches in Ch.4. Through analysis, we can get a better understanding of the important characteristics of workload that affects the throughput performance of each approach. Then, we simulate a more optimized version of each protocol under more realistic traffic conditions to see if the general trends of performance differences still hold. Simulation also allows us to evaluate the delay performance of each approach much more easily than through analysis. This understanding of the intrinsic strengths and weaknesses of various design approaches will help guide the design of a new multi-channel MAC protocol that performs well under a wide variety of conditions.

3.3 Proposing a New Solution

In Ch.5, we use the insights from the analysis and simulation of the previous chapter to drive the design of a new multi-channel MAC protocol. In designing of this protocol, we use two guiding principles extensively: *generality over optimality* and *robustness via randomization*.

The principle of *generality over optimality* says that it is better to design a system which

performs well over a wide variety of conditions than one that performs extremely well only in certain situations when the application scenario is not known precisely. The rationale behind this principle is that a MAC protocol is a fundamental protocol upon which different types of applications may build. If an application scenario is known precisely, it is often possible to further improve the performance of the MAC protocol by finding and exploiting special properties of the scenario. Once a protocol has been optimized for known scenarios, it can be too complicated to adapt to a specific application for further gains. Following the principle of generality over optimality, we made the following decisions:

- **Multi-hop Compatibility** Even though we only consider 1-hop networks in this dissertation, we intend our design to be compatible in a multi-hop ad hoc network. We therefore avoid design choices that would make our protocol perform poorly or behave incorrectly when not all nodes are in the same neighborhood.
- **No Specific Traffic Assumptions** For the simplicity of analysis, we assume that the traffic load among all nodes are symmetric. However, we remove such assumptions in the simulation to ensure that we are not optimizing our protocol for one particular traffic pattern.
- **Simplicity over Performance** The fear of adding too much complexity into a protocol is that it will never make it to any working system because it takes too much effort to implement. We opt to keep the protocol as simple as possible without sacrificing robustness or load-balancing capability.

The principle of *robustness via randomization* suggests that we randomize key aspects of the protocol to increase robustness against random channel errors and potential deadlock among nodes. Here are a few examples of how we apply this principle to our design:

- **Random Channel Hopping** Nodes hop according to their individually chosen random hopping sequences to load-balance traffic across different channels and achieve resilience to noise on a particular channel.
- **Discovery** The backup mechanism for discovering neighbors is by simple beaconing at randomly chosen times and channels. Despite its simplicity, this randomized algorithm can discover all neighbors in a reasonable amount of time. Because we make minimal assumptions about the nodes, this backup discovery mechanism is very robust.
- **Scheduling** In our protoco, nodes use a randomized scheduler to determine the order

in which receivers are served to prevent deadlock. Using a deterministic scheduler, two nodes trying to find each other persistently can easily lead to a deadlock.

3.4 Implementation of a Prototype

In Ch.6, we present a prototype implemention of our proposed protocol. The goal of the implementing a prototype is three fold:

1. **Prove Feasibility** We hope to prove that the Parallel Rendezvous approach is indeed practical and robust.

2. **Understand Complexity** We also hope to discover potential problems ignored in analysis (for tractability) or neglected in simulation through this implementation exercise. Implementation forces us to consider all the protocol details and weigh this added complexity against the benefits of using a multi-channel MAC protocol.

3. **Gain Hardware Design Insights** An implementation will allow us to identify important features of the hardware platform necessary to achieve good performance. These will serve as guidelines when designing future radio specifically for multi-channel MAC protocols.

The optimization of the various algorithm parameters necessary to achieve the best performance in an actual system is not among our objectives.

Evaluation of MAC protocols through an real implementation is often very difficult due to the lack of a suitable hardware platform. MAC protocols have hard real-time requirements on the order of micro-seconds which can only be satisfied if the protocol logics are mostly implemented in hardware rather than in software. As a result, very few platforms can provide both speed and flexibility, and all of them are very time consuming to program.

Luckily, since our major goals are to discover potential implementation difficulties and prove feasibility, a slower but flexible platform is sufficient. Based on our protocol requirements, we first evaluate many potential hardware platforms and then choose one that best fits our needs. Next, we run micro-benchmarks to measure the performance of the platform. The results allow us to determine the right parameters to be used with the newly designed protocol. For example, how often a node hops channel is dependent on the channel switching time of the radio. We select a subset of the most salient features of the protocol

that are most likely to present implementation challenges, and implement them. Finally, we measure the performance of our implementation and reflect on our experience of the implementation exercise.

Chapter 4

Comparison of Multi-Channel MAC Approaches

4.1 Introduction

Researchers have proposed protocols that exploit the multiple channels available in 802.11 and other wireless networks to increase capacity. These protocols are for networks in which orthogonal channels, such as disjoint frequency bands, are available. Using a multichannel MAC protocol, different devices can transmit in parallel on distinct channels. The parallelism increases the throughput and can potentially reduce the delay, provided that the channel access time is not excessive. Protocols differ in how devices agree on the channel to be used for transmission and how they resolve potential contention for a channel. These choices affect the delay and throughput characteristics of the protocol.

There have been only limited studies comparing these protocols under identical operating conditions. In this paper, we compare several existing and one new multi-channel MAC protocols. As may be expected, different protocols are preferable depending on the operating conditions. Our objective is to contribute to the understanding of the relative merits of different designs. We first use analytical models to gain insight into the strengths and weaknesses of the various protocols. We then use simulations to obtain more accurate comparisons assuming more realistic traffic patterns. We consider only the case of a single collision domain where all the devices can hear one another.

In Section 4.2, we describe the protocols that we compare in this paper. Section 4.3 presents simplified analytical models of the protocols. Section 4.4 discusses the numerical results that result from the analytical models and compares the performance of the protocols. Section 4.6 describes results of simulations of more realistic models of the protocols. Section 4.7 concludes the paper with some comments about the lessons of this study.

4.2 Description of Protocols

There are many variations on multichannel protocols. Our first step is to classify them based on their general principles of operation. We then describe representative protocols of the different classes. We also comment on the many variations that have been proposed for such protocols.

4.2.1 Principles of Operation

Devices using a multichannel MAC protocol exchange control information in order to agree on the channel for transmitting data. In *single rendezvous* protocols, the exchange of control information occurs on only one channel at any time. That single control channel can become the bottleneck under some operating conditions. *Multiple rendezvous* protocols allow multiple devices to use several channels in parallel to exchange control information and make new agreements. This approach alleviates the rendezvous channel congestion problem but raises the challenge of ensuring the idle transmitter and receiver visit the same rendezvous channel.

In the subsequent sections, we describe the following protocols and their variations:

1. **Dedicated Control Channel** (single rendezvous using 2 radios): devices use one radio to constantly monitor the control channel.
2. **Common Hopping** (single rendezvous using 1 radio): devices hop together quickly and stop upon agreement for transmission.
3. **Split Phase** (single rendezvous using 1 radio): devices periodically tune to the control channel together.
4. **McMAC** (multiple rendezvous using 1 radio): transmitter jumps to receiver's slow hopping sequence.

4.2.2 Dedicated Control Channel

Every device has two radios. One radio is tuned to a channel dedicated to control messages; the other radio can tune to any other channel. In principle, all devices can overhear all the agreements made by other devices, even during data exchange. This system's efficiency is limited only by the contention for the control channel and the number of available data channels.

Fig. 4.1(a) illustrates the operations of Dedicated Control Channel. In the figure, channel 0 is the control channel and channels 1, 2, and 3 are for data transmission. When device A wants to send to device B, it transmits an RTS (request-to-send) packet on the control channel. That RTS specifies the lowest-numbered free channel. Upon receiving the RTS, B responds with a CTS (clear-to-send) packet on the control channel, confirming the data channel suggested by A. The RTS and CTS packets also contain a Network Allocation Vector (NAV) field, as in 802.11, to inform other devices of the duration for which the sender, the receiver, and the chosen data channel are busy. Since all devices listen to the control channel at all times, they can keep track of the busy status of other devices and channels even during data exchange. Devices avoid busy channels when selecting a data channel.

Examples of this approach include DCA (Dynamic Channel Allocation) [32], DCA-PC (Dynamic Channel Allocation with Power Control) [33] and DPC (Dynamic Private Channel) [9].

The major advantage of Dedicated Control Channel is that it does not require time synchronization–rendezvous always happen on the same channel. The disadvantage of this protocol is that it requires a separate control radio and a dedicated channel, increasing cost and decreasing spectral efficiency when few channels are available.

4.2.3 Common Hopping

In this approach, devices have only one radio. Devices not exchanging data cycle through all channels synchronously. A pair of devices stop hopping as soon as they make an agreement for transmission and rejoin the common hopping pattern subsequently after transmission ends.

The Common Hopping protocol improves on Dedicated Control Channel in two respects: 1) it use all the channels for data exchange; 2) it requires only one transceiver per device. As shown in Fig. 4.1 (b), the hopping pattern cycles through channels 0, 1, 2 and 3. When

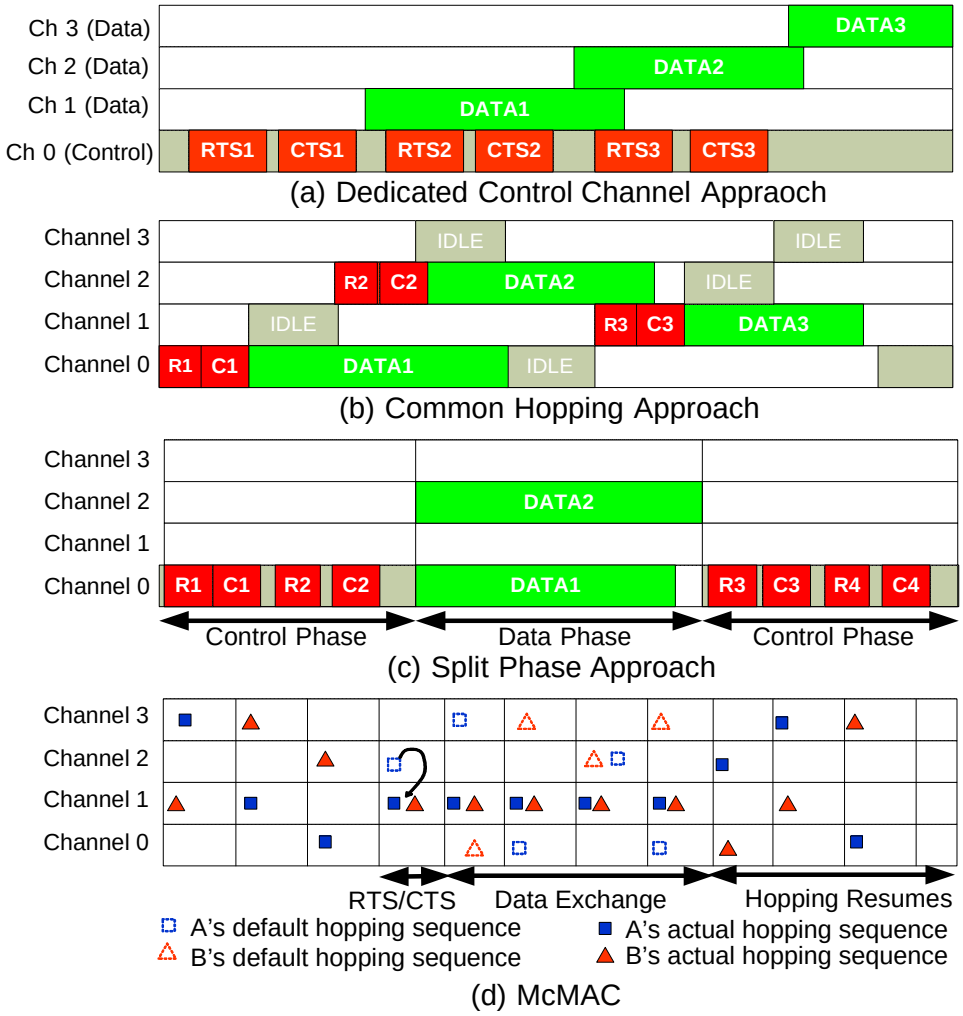


Figure 4.1. Basic operations of different MAC approaches.

device A wants to send to device B, it sends an RTS to B on the current common channel. If B receives the RTS properly, it returns a CTS on the same channel. Devices A and B then pause hopping and remain on the same channel during data transfer while the other idle devices continue hopping. When they are finished, devices A and B rejoin the common hopping sequence with all the other idle devices. It is possible that the common hopping sequence wraps around and visits the channel A and B are using before they finish data exchange. Idle devices sense the carrier and refrain from transmitting if it is busy.

While A and B are exchanging data, they are unaware of the busy status of the other devices. Hence, it is possible that a sender sends an RTS to a device that is currently busy on a different channel. Another issue with this approach is that devices hop more frequently. State-of-the-art integrated circuits implementations of tri-mode 802.11a/b/g

radios require only about 30μsec for its voltage-controlled oscillator (VCO) to settle [34], but commercial off-the-shelf 802.11b transceivers require about 150 to 200μsec to switch channels [17]. Considering that an RTS in 802.11b takes only about $200 - 300\mu$sec, the hopping time penalty is not negligible. The approach also requires devices to have tight synchronization. Examples of this design approach include CHMA (channel hoping multiple access) [31] and CHAT (Channel Hopping multiple Access with packet Trains) [30].

4.2.4 Split Phase

In this approach, devices use a single radio. Time is divided into an alternating sequence of control and data exchange phases, as shown in Fig. 4.1 (c). During a control phase, all devices tune to the control channel and attempt to make agreements for channels to be used during the following data exchange phase.

If device A has some data to send to device B, it sends a packet to B on the control channel with the ID of the lowest numbered idle channel, say, i. Device B then returns a confirmation packet to A. At this point, A and B have agreed to use channel i in the upcoming data phase. Once committed, a device cannot accept other agreements that conflict with earlier agreements. (Note that when hidden nodes are prevalent, the sender and the receiver might have a very different view of which channels are free. A more sophisticated agreement protocol is then needed, as proposed in [25].)

In the second phase, devices tune to the agreed channel and transfer data. The protocol allows multiple pairs to choose the same channel because each pair might not have enough data to use up the entire data phase. As a result, the different pairs must either schedule themselves or contend during the data phase. In the analysis, we assume that at most one device pair can be assigned to each channel, so there is no need for scheduling or contention. In the simulations, we assume random access as suggested in MMAC [25].

The advantage of this approach is that it requires only one radio per device. However, it requires time synchronization among all devices, though the synchronization can be looser than in Common Hopping because devices hop less frequently. Examples of this approach are MMAC [25] and MAP (Multichannel Access Protocol) [6]. Their main difference is that the duration of the data phase is fixed in MMAC whereas it is variable in MAP and depends on the agreements made during the control phase.

4.2.5 Multiple Rendezvous

Multiple rendezvous protocols differ from the previous three in that multiple device pairs can make agreements simultaneously on distinct channels. The main motivation is to overcome the single control channel bottleneck. However, since there are multiple rendezvous channels, special coordination is required so that two devices can rendezvous on the same channel. One solution is for each idle device to follow a "home" hopping sequence and for the sending device to transmit on that channel to find the intended receiver. Examples of this approach include SSCH (Slotted Seeded Channel Hopping) [5] and McMAC [23].

In SSCH there are as many hopping sequences that each device can follow as there are channels. Each sequence is uniquely determined by the seed of a pseudo-random generator. Each device picks multiple (e.g., 4) sequences and follows them in a time-multiplexed manner. When device A wants to talk to B, A waits until it is on the same channel as B. If A frequently wants to talk to B, A adopts one or more of B's sequences, thereby increasing the time they spend on the same channel. For this mechanism to work, the sender learns the receiver's current sequences via a seed broadcast mechanism.

In McMAC, each device picks a seed to generate an independent pseudo-random hopping sequence. When a device is idle, it follows its default hopping sequence as shown in Fig. 4.1 (d). Each device puts its seed in every packet it sends, so its neighbors eventually learn its hopping sequence. For simplicity, devices are assumed to hop synchronously. The hopping can be made less frequent than Common Hopping to reduce the channel switching and synchronization overhead. When device A has data to send to B, A flips a coin and transmits with some probability p during each time slot. If it decides to transmit, it tunes to the current channel of B (e.g., channel 1) and sends an RTS. If B does not reply with a CTS, either due to an error or because B is busy, then A tries again later by coin flips. If B replies with a CTS, both A and B stop hopping to exchange data. Data exchange normally takes several time slots. After the data exchange is over, A and B return to their original hopping sequence as if there was no pause in hopping.

SSCH and McMAC are similar in that they allow devices to rendezvous simultaneously on different channels. In the rest of the paper, we focus on McMAC as an example protocol in this category.

4.2.6 Protocol Variations

In this paper, we compare four generalized approaches to designing multi-channel MAC protocols. An actual protocol includes fine adjustments that deviate from the generalized scheme. Instead of incorporating all possible variations of the four schemes in our analysis and simulation, we briefly mention several proposed improvements and discuss their effects qualitatively.

For Dedicated Control Channel, it is possible to use the control channel for data transfer when all other channels are busy. For Split Phase, adaptation of the duration of data and control phases was proposed by [6]. [24] suggests advertising the number of packets for each destination in the rendezvous message to achieve better load balancing across channels.

4.3 System Model and Analysis

In this section, we describe and analyze simplified models of the protocols. The objective of the models is to enable numerical comparisons of the performance characteristics of the protocols under identical operating conditions. The following simplifications are made for all the protocols:

1. Time is divided into small slots, with perfect synchronization at slot boundaries;
2. Nodes contend for medium using the ALOHA protocol. Under this model, each node wanting to transmit flips a coin during each time slot. With some probability p_{tx}, it will transmit. If exactly 1 node transmits, the packet is successfully received by the receiver. Otherwise, collision occurs and the slot is wasted.
3. Upon making an agreement, the devices can transmit only one packet (one may think of a "packet" as the amount of data that can be transferred per channel agreement);
4. The packet lengths are independent and geometrically distributed with parameter q (i.e., with mean $1/q$);
5. Devices always have packets to send to all the other devices; in each time slot, an idle device attempts to transmit with probability p.

The first simplification enables us to use discrete time models. The second one allows for a simple model to account for the overhead of contention. The third one reduces the

efficiency of all the protocols since it requires a new agreement for every packet. The fourth simplification is needed to construct simple Markov models. We use the fifth simplification to identify the throughput of the protocols.

These simplifications allow us to form a Markov chain whose state X_t is the number of communicating pairs at time t. When $X_t = k$, $2k$ devices are involved in data transfer while the other $N - 2k$ devices are idle. The maximum number of pairs is bounded by the number of M_D of data channels and $\lfloor N/2 \rfloor$, half the number of devices. Accordingly, the state space of X_t is

$$\mathcal{S} := \{0, 1, \ldots, \min(\lfloor \frac{N}{2} \rfloor, M_D)\}.$$

The number M_D of data channels is equal to M for all approaches except Dedicated Control Channel for which $M_D = M - 1$ since a channel is reserved for control.

A state transition happens when new agreements are made or existing transfers end. Let $S_k^{(i)}$ and $T_k^{(j)}$ respectively denote the probability that i new agreements are made and the probability that j transfers terminate in the next slot when the state is k. The state transition probability p_{kl} from state k at time t to l at time $t+1$ can be expressed as follows:

$$p_{kl} = \sum_{m=(k-l)^+}^{k} S_k^{(m+l-k)} T_k^{(m)}. \tag{4.1}$$

In this expression, m is the number of transfers that terminate and its value is between $(k-l)^+$ and k. At least $(k-l)^+$ transfers should terminate to have l pairs in the next slot and k is the maximum number of terminating transfers. Also, the probability $T_k^{(j)}$ that i transfers finish when the system is in state k is given by the following expression:

$$\begin{aligned} T_k^{(j)} &= Pr[j \text{ transfers terminate at time } t | X_{t-1} = k] \\ &= \binom{k}{j} q^j (1-q)^{k-j}. \end{aligned} \tag{4.2}$$

Equation (4.1) is further simplified in the single rendezvous protocols such as Dedicated Control Channel or Common Hopping because $S_k^{(i)} = 0$ for all $i > 1$. Indeed, at most one additional pair can meet in the next slot in a single rendezvous protocol. Accordingly, for such protocols, the equation becomes

$$p_{kl} = T_k^{(k-l)} S_k^{(0)} + T_k^{(k-l+1)} S_k^{(1)} \tag{4.3}$$

where $T_k^{(j)} = 0$ when $j < 0$.

The average utilization ρ per channel can be obtained as

$$\rho = \frac{\sum_{i \in \mathcal{S}} i \cdot \pi_i}{M} \tag{4.4}$$

where π_i is the limiting probability that the system is in state i and $\mathcal{S}$ is the state space of the Markov chain. One obtains π_i by solving the balance equations of the Markov chain. We then derive the total system throughput by multiplying ρ by the channel transmission rate and by M_D, the number of data channels.

4.3.1 Dedicated Control Channel

Devices constantly monitor the control channel and keep track of which devices and data channels are idle. When a device has packets to transmit to an idle receiver, it sends an RTS message for that idle receiver on the control channel. If it hears the RTS, the receiver replies to the sender with a CTS. Then both the sender and the receiver tune to the agreed channel to start transmission.

An agreement is made when exactly one idle device attempts to transmit an RTS on the control channel. Hence, the success probability $S_k^{(i)}$ in the next time slot, given that k pairs are communicating in the current slot, is:

$$S_k^{(i)} = \begin{cases} (N-2k)p(1-p)^{(N-2k-1)}, & \text{if } i = 1; \\ 1 - S_k^{(1)}, & \text{if } i = 0; \\ 0, & \text{otherwise.} \end{cases} \tag{4.5}$$

The transition probabilities (4.3) can be rewritten as follows:

$$p_{kl} = \begin{cases} 0 & \text{if } l > k+1 \\ T_k^{(0)} S_k^{(1)}, & \text{if } l = k+1; \\ T_k^{(k-l)} S_k^{(0)} + T_k^{(k-l+1)} S_k^{(1)}, & \text{if } 0 < l \leq k; \\ T_k^{(k)} S_k^{(0)} & \text{if } l = 0. \end{cases} \tag{4.6}$$

We obtain the system throughput R_d as

$$R_d = (M-1)C\rho_d \tag{4.7}$$

where C is the channel capacity and ρ_d is the data channel utilization that we calculate using (4.4); the subscript d refers to the Dedicated Control Channel approach.

4.3.2 Common Hopping

The analysis of this protocol is very similar to that of the Dedicated Control Channel, but with three differences: 1) devices do not track the status of each other; 2) some slots are

busy and unavailable for control messages; 3) the switching penalty is incurred whenever a device hops.

Information about Other Devices

Even if an RTS is sent without collision, the receiver may not respond because it might be busy on a different channel. When a device is sending or receiving, it cannot keep track of others. At best, idle devices can keep track of the agreements that others make. However, this information becomes stale once the devices become busy. We approximate this situation by assuming that the sender selects the receiver uniformly out of $N-1$ other devices and that the probability that the selected receiver is not busy is $\frac{N-2k-1}{N-1}$.

Busy Slots

The protocol can make a new agreement only when the current common hopping channel is idle. We model this effect by considering that the probability that idle devices can use a given slot to make an agreement is $(M-k)/M$ when k channels are busy.

Combining the effects described in the previous two points, the success probability $S_k^{(i)}$ now becomes:

$$S_k^{(i)} = \begin{cases} (N-2k)p(1-p)^{(N-2k-1)}\frac{N-2k-1}{N-1}\frac{M-k}{M}, & \text{if } i=1; \\ 1-S_k^{(1)}, & \text{if } i=0; \\ 0, & \text{otherwise.} \end{cases} \tag{4.8}$$

The factor $\frac{N-2k-1}{N-1}$ in (4.8) is the probability that the receiver is one of the $N-2k-1$ idle devices other than the transmitter among the other $N-1$ devices. The last term $\frac{M-k}{M}$ is introduced to represent the chance that the common hopping channel is busy when k channels are busy.

Channel Switching Penalty

Let t_s and t_p denote the duration of one slot and the channel switching penality respectively. One slot in the Common Hopping protocol has duration $t_s' = t_s + t_p$. Since the hopping penalty does not occur when devices transmit data in the other schemes, we can think of each slot as being slightly longer in Common Hopping, which results in a higher packet termination probability q. To make a fair comparison, one should choose

$q' = q \cdot \frac{t_s + t_p}{t_s}$. The system throughput R_c is as follows:

$$R_c = MC\rho_c \tag{4.9}$$

where C is the channel capacity and ρ_c is the utilization of the system that can be calculated using (4.4). The subscript c denotes Common Hopping. Note that the multiplier is M instead of $M - 1$ in Dedicated Channel.

4.3.3 Multiple Rendezvous

We analyze McMAC [23] in this section. In McMAC, each device has a home pseudo-random hopping sequence defined by a seed. Eventually, through broadcast, every device knows the seed of the other devices and can determine their home channel at any given time.

In McMAC, a device is unaware of which devices and channels are idle. We assume that any idle device is equally likely to have any channel as its current home channel and that these home channels are independent. When a device is idle, it attempts to transmit in the next time slot with probability p. In that case, the device chooses another device at random and goes to its channel.

To make j new agreements the following conditions must hold:

(1) The number A of devices that attempt to transmit should be at least j.

(2) The devices attempting to transmit should be on a channel without other attempts. Let O denote the number of channels where those isolated attempts take place; we call these channels "one-attempt channels."

(3) The channel where a device attempts to transmit should be idle. We designate by I the number of idle channels among the O one-attempt channels.

(4) Designate by J the number of transmitters out of I who can find a receiver that is idle and does not attempts to transmit. Then $J = j$.

We compute the probability $S_k^{(i)}$ by conditioning on the values of O, I and A. We can write

$$S_k^{(j)} = \sum_{o,i,a} P[A = a]P[O = o|A = a]P[I = i|O = o, A = a]$$
$$\times P[J = j|I = i, A = a, O = o]$$

- $P[A = a] = \binom{N-2k}{a} p^a (1-p)^{(N-2k-a)}$ for $a \geq j$;

- $P[O = o | A = a]$ is the probability that o devices out of a are in a channel without other attempts given that a devices attempt. This probability is the same as the probability that o urns out of M contain exactly one ball after we throw a balls independently and uniformly in M urns. If we let Y_i^n be the number of urns with i balls at the n-th throw, then (Y_0^n, Y_1^n) is a Markov chain and the probability distribution of Y_1^a can be computed recursively over $n = 1, \ldots, a$ from the transition probabilities of that Markov chain.

- $P[I = i | O = o, A = a]$ is the probability that i channels out of o one-attempt channels are idle, which is equivalent to the probability that exactly $o - i$ out of o one-attempt channels are busy. Recall that k out of M channels are busy. We pick k busy channels out of M uniformly and also pick o one-attempt channels out of M uniformly, independently of each other. The intersection of the two sets corresponds to busy one-attempt channels. The conditional distribution of I is given by

$$P[I = i | O = o, A = a] = P[I = i | O = o] = \frac{\binom{k}{o-i}\binom{M-k}{i}}{\binom{M}{k}}.$$

  The first equality comes from that I and A are conditionally independent given O. The denominator is the number of ways to select k busy channel out of M. The numerator is the number of ways to select $o - i$ one-attempt channels among k busy ones and to select i one-attempt channels among $M - k$ channels.

- We approximate the probability that a sender that attempts to transmit alone in an idle channel finds its receiver by $p_s = \frac{N-2k-a}{N-1}$. Moreover, we assume that the i transmitters are independently successful in finding their receivers. Let J be the number of successful senders who are alone in an idle channel and are able to find their receiver successfully. Then,

$$\begin{aligned} P[J = j | I = i, O = o, A = a] &= P[J = j | I = i, A = a] \\ &= \binom{i}{j} p_s^j (1 - p_s)^{(i-j)}. \end{aligned}$$

4.3.4 Split-Phase Approach

In Split-Phase, time is divided into alternate control and data phases with durations c and d, respectively. Let $R(n)$ be the throughput of the n-th period. By ergodicity,

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n} R(n)}{n} \to R_s$$

where R_s is the expected throughput per period of the Split Phase approach.

We designate by K_n the random number of agreements made during the n-th control phase and we define

$$\phi_i^c := P(K_n = i), \text{ for } i = 0, 1, 2, \ldots, \gamma := \min\left(c, \lfloor \frac{N}{2} \rfloor\right). \tag{4.10}$$

The above probability can be computed using the following recursive relationship on the duration c:

$$\phi_i^c = s_{i-1}\phi_{i-1}^{c-1} + (1 - s_i)\phi_i^{c-1}, \tag{4.11}$$

where s_i is the probability that an agreement is made in a slot when i pairs of nodes have already made an agreement. Nodes who have already made an agreement do not contend for medium. Hence, $s_i = (N - 2i)p_{tx}(1 - p_{tx})^{N-2i-1}$ where p_{tx} is the probability of a node deciding to transmit in each slot according to the ALOHA model. Note that $\phi_0^c = (1 - s_0)^c$ and $\phi_i^c = 0$ when $i > c$.

Let $R(i, d)$ denote the average throughput in the duration of d when i agreements are made in the control phase. Then the channel utilization ρ_s can be computed using the following equation:

$$\rho_s = \frac{1}{(c + d)M} \sum_{i=1}^{\gamma} \phi_i^c R(i, d). \tag{4.12}$$

The system throughput R_s is

$$R_s = MC\rho_s \tag{4.13}$$

where C is the channel capacity.

The throughput $R(i, d)$ when i new agreements are made in the control channel cannot be larger than Md. If we assume perfect distribution of i agreements over M channels, each channel has either l or $l+1$ agreements where $l := \lfloor \frac{i}{M} \rfloor$. Since the packet size is geometrically distributed, the channel that has l agreements can be utilized up to $\min\left(\sum_{j=1}^{l} Y_j, d\right)$ slots where the $Y_j's$ are geometrically distributed random variables. Therefore, the averaged throughput can be upper-bounded as follows:

$$R(i,d) \leq (M-r)E\left[\min\left(\sum_{j=1}^{l} Y_j, d\right)\right] + rE\left[\min\left(\sum_{j=1}^{l+1} Y_j, d\right)\right] \tag{4.14}$$

where r is the number of channels with $l+1$ agreements. Thus $(M-r)$ is the number of channels with l agreements.

Let W_l be $\sum_{j=1}^{l} Y_j$. By conditioning on $Pr(W_l \leq d)$,

$$E[\min(W_l, d)] = d(1 - Pr(W_l \leq d)) + E[W_l | W_l \leq d] Pr(W_l \leq d) \tag{4.15}$$

One can then calculate (4.15) numerically using an appropriate negative binomial distribution.

The right side of (4.14) is an upper bound since we assume perfect alignments among l pairs or $l+1$ pairs in each channel. The real throughput can be smaller due to collision among l pairs. However, we can use the right side as an approximation when the value l is small.

If c is too short, the multiple channels may not be utilized; On the other hand, if c is too long, the packet may suffer the long delay and under utilization. Therefore, choosing appropriate values for c and d is crucial to the performance of this scheme.

It is possible for multiple pairs of nodes to be assigned to the same channel because there are fewer channels than the number of successful sender/receiver pairs in the control phase. Therefore, in addition to the contention in the control phase, there is also competition for medium in the data phase. In MMAC[25], senders contends for medium in the data phase using RTS/CTS. To simplify the analysis, however, we assume that the different pairs use the medium completely efficiently in the data phse without wasting any time due to contention or control packet overhead.

4.4 Numerical Results

In this section, we compare the different protocols using the analytical models that we have described so far. Several approximations were made in the analysis presented in the previous section. To verify that such approximations are reasonable, we compared the analytical results to Monte Carlo simulations of the protocol models. We call this set of simulations SimLite to distinguish them from more sophisticated simulation scenarios to

be introduced later. We also vary the parameters such as the number of channels, channel switching time, number of devices to observe their effects on the throughput.

We created two simulation scenarios: 802.11b and 802.11a. In the 802.11b scenario, there are 20 devices, 3 channels, and each channel has a data rate of 2 Mbps. In the 802.11a scenario, there are 40 devices, 12 channels with a rate of 6 Mbps each. The time it takes for an RTS/CTS exchange are $812\mu s$ and $200\mu s$ respectively in 802.11b and 802.11a scenarios. The channel switching time is $100\mu s$ by default. For Split Phase, the default control phase/ data phase durations are 20ms/40ms for 802.11b and 10ms/10ms for 802.11b. The packets have a random length with a geometric distribution with average length of 1 KB or 10 KB. Since there is no queueing, one should think of the packet length as the amount of data a device can transfer after each channel agreement. Nonetheless, we will continue to use the term *packet length* throughout this section. Fig. 4.2 and Fig. 4.3 show that the analytical throughput models of all 4 approaches are in close agreement with SimLite for both 802.11b and 802.11a scenarios. This shows that the approximations we made in our analysis are reasonable.

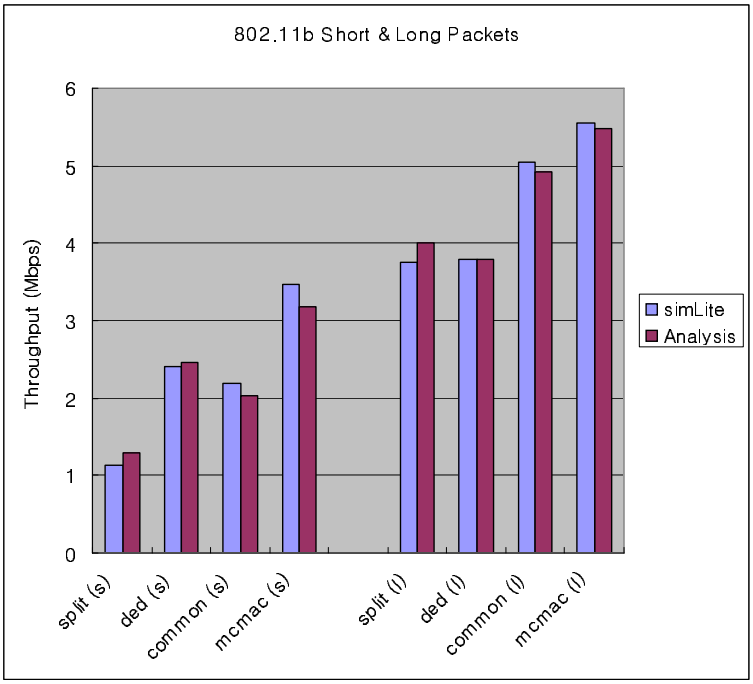


Figure 4.2. Throughput predicated by analysis vs. simulation (SimLite) under 802.11b settings.

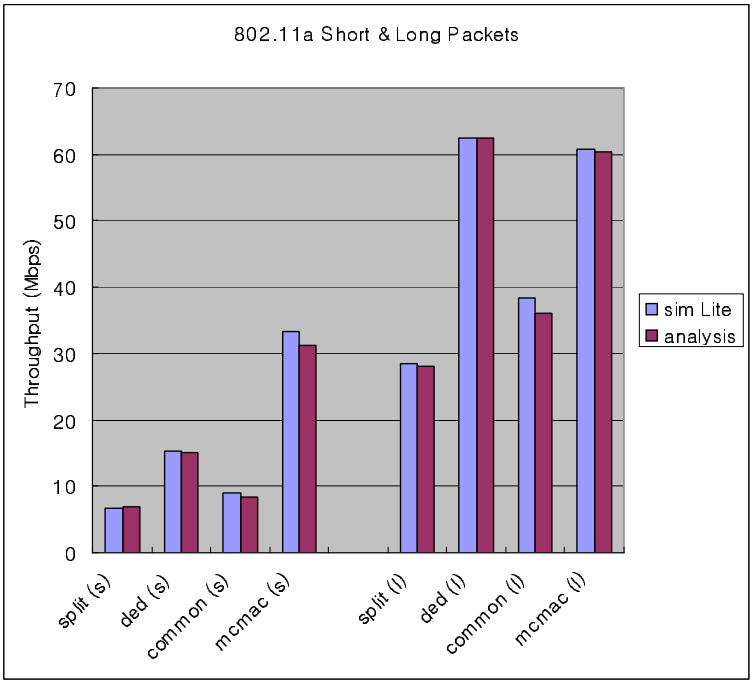


Figure 4.3. Throughput predicated by analysis vs. simulation (SimLite) under 802.11a settings.

4.4.1 802.11b Scenario

Fig. 4.2 shows the throughput results of four different protocols under the 802.11b scenario.

Dedicated Channel Protocol. The protocol can use only two channels out of three for data which limits its maximum throughput to 4 Mbps which is much less than Common Hopping or McMAC. Given that constraint, however, Dedicated Control Channel still achieves 3.8Mbps.

Parallel Rendezvous Although the control channel is not saturated, McMAC still performs the best for both short and long packets because (i) it uses all 3 channels for data transfer, and (ii) it does not have any quantization overhead of Common Hopping since it can reuse a channel as soon as the previous transfer has finished. Common Hopping cannot reuse a channel until the common hopping sequence has wrapped around to it. For long packets, McMAC achieves a remarkable 5.5 Mbps out of 6 Mbps.

Split-phase Approach. The throughput of Split Phase is the lowest for 1 KB packets and close to the lowest for 10 KB packets even though we have optimized the durations of the control and data phases (i.e., c and d) to be 20 msec and 40 msec.

4.4.2 802.11a Scenario

The bar chart on the right of Fig. 4.3 shows the throughput results of the four different protocols under the 802.11a scenario.

Parallel Rendezvous. McMAC achieves about 31Mbps in the small packet size case (left) and 60Mbps for the large packet size case (right). Note that Dedicated Control Channel performs slightly better than McMAC when packets are large, but recall that it uses two radios.

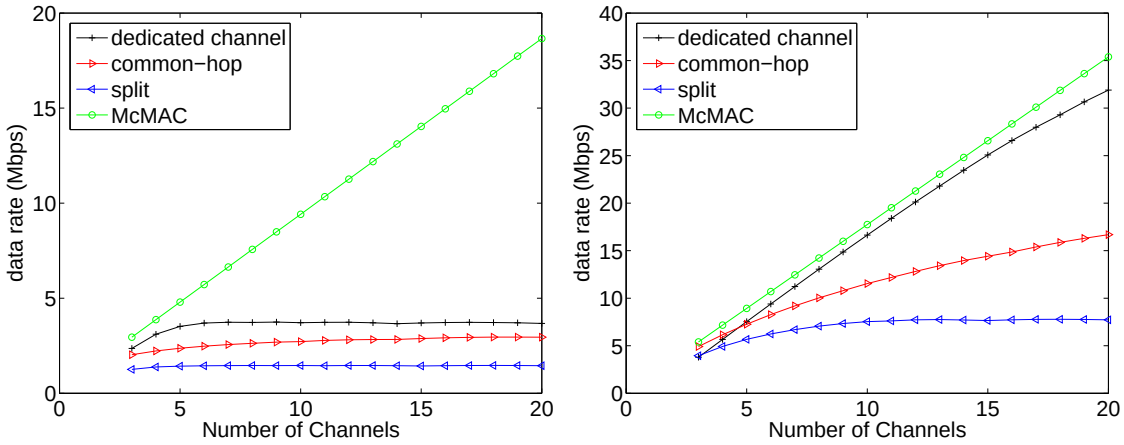
Control Channel Congestion. In single rendezvous protocols where there is only one control channel, a combination of short packets and a large number of channels can cause control channel congestion as shown on Fig. 4.3. The best possible throughput for single rendezvous channel protocols can be estimated as follows. On average, one agreement is made every $1/P_{succ} \approx e = 2.718$ slots. After each agreement, a 1 KB packet is transferred on average. Since it takes 6.8 slots to transfer 1 KB packet (200μ sec per slot), the maximum throughput achievable by any single rendezvous protocol under these assumptions is $(6.8/e) \times 6\text{Mbps} \approx 15\text{Mbps}$. Dedicated Control Channel, Common Hopping and Split Phase achieve 15Mbps, 9Mbps, and 8Mbps respectively.

Dedicated Channel with Long Packets. Observe that Dedicated Control Channel achieves more than 63Mbps when the average packet size is 10Kbyte. With a large average packet size, the control channel no longer is a bottleneck because for the same amount of data traffic, the number of channel agreements required is greatly reduced. Moreover, when 12 channels are available, using 1 of them for control traffic is a small overhead to pay.

4.4.3 Scaling Number of Channels

Fig. 4.4 shows the throughput (y-axis) when using different numbers of channels (x-axis) for two average packet sizes. In order to ensure that we have enough number of nodes to effectively use all the available channels, we set the number of devices in each simulation to be 6 times the number of channels. We assumed that the bandwidth per channel is 2Mbps. The slot time is equal to 812μs while the channel switching time is 100μs. For Split Phase, we assume the control and data durations to be 20 msec and 40 msec respectively.

McMAC scales well with an increasing number of channels while the single rendezvous protocols do not. In Fig. 4.4(a) where packets are smaller, the throughput of all the single channel protocols (Split Phase, Dedicated Control Channel, Common Hopping) flatten out



(a) (b)

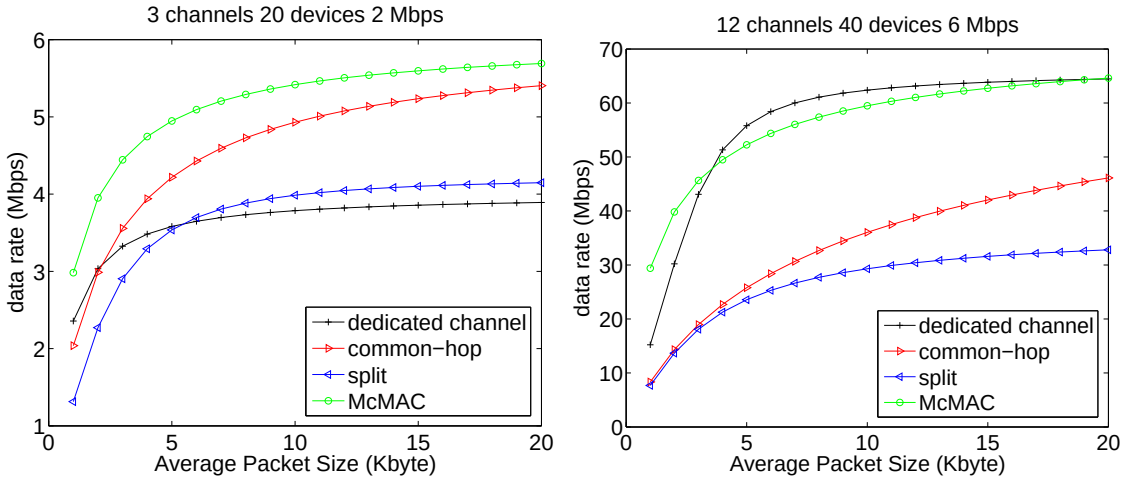
Figure 4.4. Throughput vs. number of channels. Average packet size is 1Kbyte (a) or 10Kbytes (b).

when the number of channels is more than 6 because of the congestion on the control channel. With longer packets, the flattening of the curves starts to occurs later. This shows that the single rendezvous protocols can use more channels efficiently in such case. However, the bottleneck of the control channel is only delayed, but *not* avoided. As we increase the number of channels, eventually, the single control channel will become the bottleneck and limit the throughput of the system.

The simulation results of [24] (Fig. 11) show that the performance of Split Phase is better than Dedicated Control Channel as the number of channel increases. The authors compare MMAC, a Split Phase algorithm with a Dedicated Control Channel approach, DCA, with 3 to 6 channels. The aggregate throughput of MMAC with 6 channels was about 3.7Mbps while that of the DCA was 2.3Mbps. The overall trend of their results are opposite to that of our findings shown in Fig.4.4. The discrepancy stems mainly from two differences. First, in the channel scaling study of [24] (Fig. 11), the authors assume a sender can send multiple packets to the same receiver within each data phase due to queueing. Second, they assume that the traffic among different device pairs are disjoint, such that each receiver will only communicate with one sender. When the traffic is non-disjoint but queueing is still present, [24] shows the performance gap between the two approaches narrows.

4.4.4 Sensitivity to Packet Size

Fig. 4.5 shows the performance with different average packet sizes (x-axis) when the number of channels is 3 (left) and 12 (right). The throughput of all schemes increases with the size of packets which can be explained by the increase in the data transfer duration after each channel agreement. Observe in the left plot (3 channel case) that the Dedicated Control Channel becomes worse than Split Phase with packet sizes larger than 5KB. Dedicated Control Channel is bounded by 4Mbps since one channel is used for control messages. The performance of Split Phase depends very much on the packet sizes. Its throughput increase from 1.2Mbps with 1Kbyte packet to 3.5Mbps with 5Kbyte packet. It is recommended that the Split Phase transmit as much as possible when a mobile gains access to the medium to maintain reasonable channel utilization. However, even when the packet size is large, it is still worse than the others in the right plot (12 channels). Another interesting observation in the right plot is that the slope of Dedicated Control Channel is very steep when packet are short. As the control channel congestion alleviates quickly with larger packets, the throughput increases rapidly.



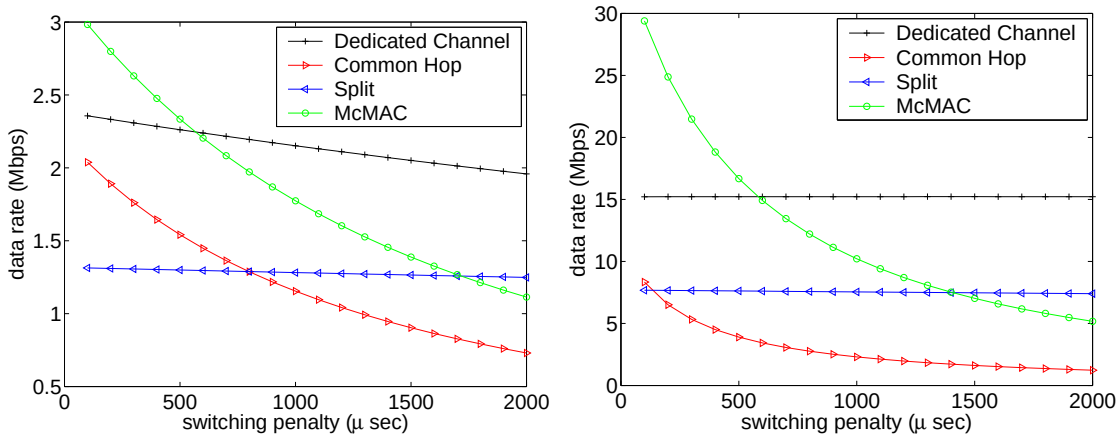
(a) (b)

Figure 4.5. Throughput vs. packet size. (a) Slot Time = 812μs. Switching Time = 100μs. 2Mbps/channel. (b) Slot Time = 200μs. Switching Time = 100μs. 6Mbps/channel.

4.4.5 Switching Penalty

Fig. 4.6 shows the performance with increasing switching penalty from 100μs to 2000μs when the number of channels is 3 (left) and 12 (right), respectively. The commercial off-the-

shelf 802.11b transceivers require about 150 to 200μsec to switch channels. The throughput of Common Hopping and McMAC decrease faster whereas that of Split Phase and Dedicated Control Channel are almost insensitive to the switching time. This is because the first two approaches are based on hopping and incur a penalty every time they hop. When the switching penalty is high, Dedicated Control Channel and Split Phase are more efficient.



(a) (b)

Figure 4.6. Throughput vs. Switching Penalty. (a) Slot Time = 812μs. 2Mbps/channel. (b) Slot Time = 200μs. 6Mbps/channel.

4.5 Simulation Model

We developed a slotted-time simulator in Matlab to compare the different protocols in more realistic settings. All nodes have the same time slot boundaries. They can hear each other in this 1-hop network. Packets are lost due to collision but not to random noise.

There are three major differences between the simulation and the analytical model. The first is the existence of per-destination packet queues in each device in the simulator in order to the effects of queueing and delay. Second, the traffic models in the simulations are more realistic. Third, several protocol improvements are added in the simulations.

The assumptions and key features of the simulator are as follows:

- *Coarse-Grained Discrete Time*: The simulator works in discrete time. Each slot corresponds to the time required for making an agreement (e.g., RTS and CTS).
- *Modeling of CSMA/CA Medium Contention*: In the analytical model, we approxi-

mated the transmission success probability using the slotted ALOHA model. The carrier-sense feature in today's radios in effect reduces the penalty of a collision from hundreds of micro-seconds (packet time scales) to carrier-sense slots (micro-seconds). We introduce a parameter, the *contention success probability* p_{succ}, to model the medium contention mechanism. During each time slot, with probability p_{succ}, a winner is chosen randomly among all the devices contending for the medium on that channel. By picking appropriate values of p_{succ} we believe that this model can approximate the behavior and performance of CSMA/CA. One may argue that this model is an oversimplification of the situation. However, since this approximation is used for all models, it should result in a fair comparison.

- *Queueing of Packets*: Each device stores packets in per-destination queues. To improve performance, a sender can send multiple packets to the same destination back-to-back without contention after every packet. Once a device starts sending data, it continues until either its queue for this destination becomes empty, a preset *medium occupancy limit* is reached, or in the case of Split Phase, that the data phase has ended.

- *Traffic Models*: We use two classes of traffic: constant bit rate (CBR) which models voice or video traffic and file transfers. Traffic generation will be discussed in more detail in the upcoming Sec.4.5.1.

- *Split-Phase:* For the Split-Phase protocol, to increase performance, we allow multiple pairs to share a channel in the same data phase if no more free channels are available, as in MMAC [24]. More details can be found in Sec.4.5.2.

4.5.1 Traffic Generation

Two types of traffic are generated: CBR and file packets. The CBR traffic models voice or video flows with periodic packet arrival of a particular size. File traffic is modeled as i.i.d. packet arrivals among all node pairs. File traffic is unidirectional. During each time slot, a packet will arrive at node i for any other node j with some probability. The arrival of packets at every (i,j) pair is i.i.d. in each of the time slots. The probability of arrival is determined by the desired file traffic load, number of nodes, average packet size, and the number of channels.

CBR traffic generation is based on a simple periodic packet arrival model between

pairs of nodes who engage in bi-directional conversation. Each node talks to δ distinct communication partners. We call δ the degree of communication. At the beginning, all nodes has no CBR connections. Therefore, node 1 has δ free degrees initially. We choose δ neighbors uniformly at random and create δ CBR connections between node 1 and them. Next, we examine node 2. Node 2 has either δ or $\delta - 1$ free degrees, depending on whether node 1 has a connection to it. For each of the free degree, a distinct neighbor for which it does not already have a connection is chosen randomly. We proceed similarly to create connections for each node until either all nodes have δ distinct communication partner, or it is not possible to create any more connections because no more distinct neighbors are left. The latter can happen deterministically when the number of nodes and δ are both odd. It can also happen by chance. When it happens, the process of generating a traffic pattern restarts from scratch.

4.5.2 Improvements to Split-Phase

During the control phase, all nodes go to the first channel to meet their receivers. Senders only attempt to make a channel agreement with a receiver if not both of them are already committed to a channel in the upcoming data phase. We assume nodes can overhear all control packets in the control phase and so they know which nodes are committed. When a sender successfully finds its receiver, they choose a data channel with the fewest number of known pairs of sender/receiver. (This is different from assigning nodes pairs to channels in a round-robin fashion.)

At the beginning of the data phase, uncommitted nodes go to the last channel which is likely to be less congested. During the data phase, sender nodes have to compete for medium for a second time because multiple pairs can choose the same data channel. This contention is also modeled by picking a successful sender/receiver pair with some fixed probability as is the case in the control phase.

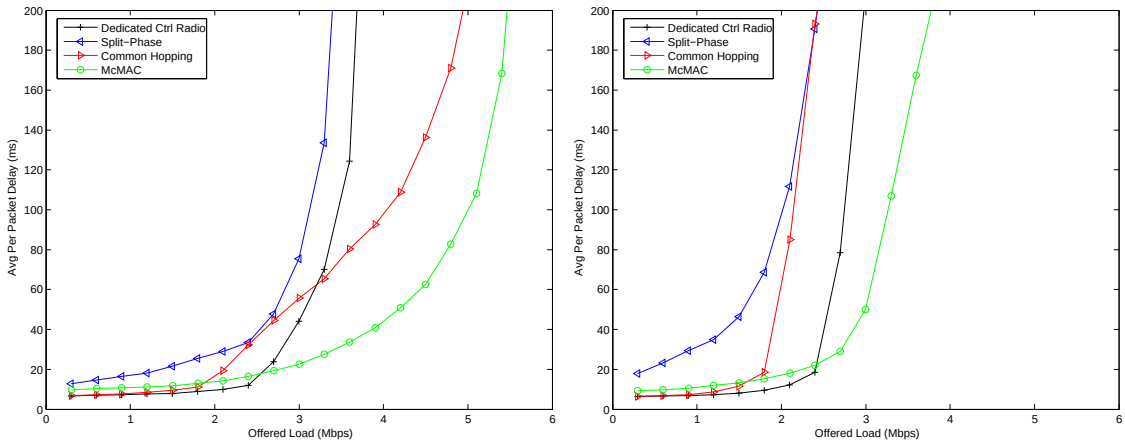
Since we assume nodes can overhear all the control packets in the control phase, they know which nodes are present in the same channel. Nodes can send packets to any other node on the same channel regardless of whether they have made an agreement or not. Once a contention is successful, the data channel is reserved for the pair until either the data phase ends or that the sender has run out of packets to send to this receiver, whichever occurs first.

4.6 Simulation Results

4.6.1 Impacts of Receiver Contention

For multi-channel MAC, the ideal traffic pattern is a disjoint one in which each sender only communicates with one receiver, and vice versa. When the traffic is non-disjoint, it is possible for a sender to pick a busy receiver, thereby wasting precious medium time. To investigate the impact of receiver contention, in this section, we vary the number of simultaneous CBR connections per device. Each device sets up CBR connections to δ distinct destinations. The first device chooses δ different destinations. The rest of the nodes choose up to δ destinations if they are involved in less than δ connections. Only destinations who have less than δ connections are eligible. A larger value of δ corresponds to more contention for receiver devices which will likely lower efficiency and increase delays.

Fig. 4.7 shows the throughput of various schemes as δ increases under the 802.11b settings. Fig. 4.8 shows the throughput under the 802.11a settings. The graphs on the left correspond to $\delta = 1$ and the right ones to $\delta = 5$. Next, we summarize our observations.

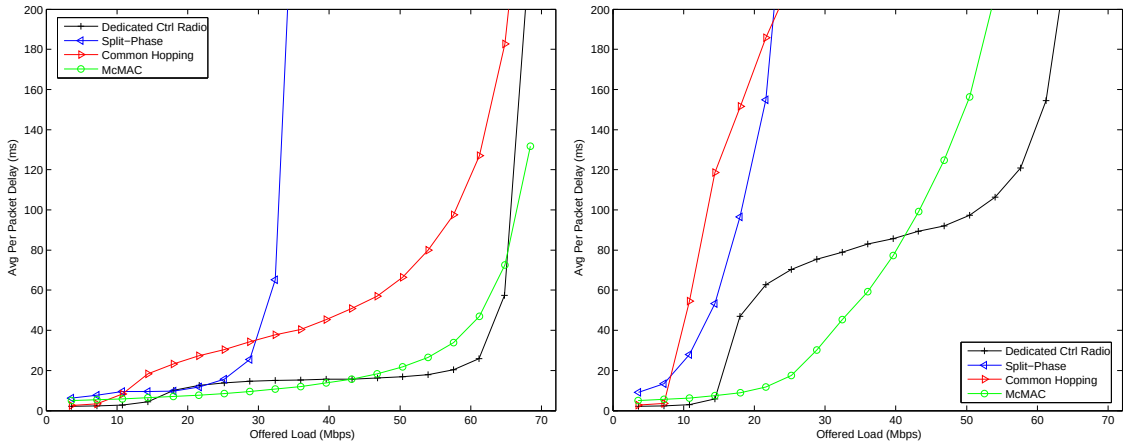


(a) (b)

Figure 4.7. Avg per packet delay vs. CBR traffic load. Left: $\delta = 1$; Right: $\delta = 5$; 802.11b settings.

4.6.2 Throughput Degradation

Given a fixed load, as δ increases, the average amount of data sent over each pair of devices decreases. Therefore, one expects a shorter communication to take place after each



(a) (b)

Figure 4.8. Avg per packet delay vs. CBR traffic load. Left: $\delta = 1$; Right: $\delta = 5$; 802.11a settings.

channel agreement. As a result, the frequency for making channel agreements increases, thereby decreasing the throughput for all schemes.

Moreover, for Split Phase, a further decrease in the throughput results as δ increases because each sender has to visit more receivers to deliver its packets. When these receivers have chosen different channels during a data phase, a sender can only deliver a fraction of packets to the subset of receivers on the same channel, resulting in further loss of efficiency.

For Common Hopping, each device cannot keep track of which other devices are busy. When $\delta = 1$, the receiver is always available when a sender successfully wins contention on the common hop since each receiver talks to only 1 sender. However, when δ is large, the probability that any particular receiver is available approaches $\frac{N-2k-1}{N-1}$ where k is the number of busy device pairs. This probability is small when the fraction of devices that are busy is large (i.e., high utilization and a small total number of devices compared to the number of channels). As a result, the last few channels are more difficult to be used efficiently.

For McMAC, the cause of degradation is similar to that of Common Hopping. However, since it allows parallel rendezvous, McMAC suffers less than Common Hopping when channel agreement traffic increases.

Finally, in Common Hopping, after a channel has been reserved by a pair of nodes, this channel cannot be reused by another pair until the common hopping sequence visits this channel again $M - 1$ slots later where M is the number of channels. In other words, the

medium can only be reserved in multiples of $M-1$ slots. When δ increases, the amount of data to transmit after each channel agreement decreases, and so a larger fraction of the $M-1$ slots is wasted. In 802.11a, 12 channels are available compared to 3 in 802.11b. This explains why the throughput degradation of Common Hopping as δ increases is more severe in the 802.11a scenario.

4.6.3 Insensitivity of Dedicated Channel

From Fig.4.7 (i.e., 802.11b scenario), we observe that Dedicated Control Channel is insensitive to the degree of communication because it knows the channel and device busy status perfectly by using a second radio. Since one of the 3 channels is set aside for control communication, the increase rendezvous traffic due to an increase in δ is easily absorbed by the abundance of control channel capacity. As the number of available channels increases to 12 in 802.11a (Fig.4.8), the cost of using an extra channel for control purposes is relatively small. The unique ability to know the channel and busy status of device precisely allows Dedicated Control Channel to achieve very good performance.

4.6.4 Delay Curves of Dedicated Channel

The delay curves of Dedicated Control Channel in Fig.4.8 are not convex under the 802.11a settings and this requires some explanation. Using Dedicated Control Channel, a pair of devices must first rendezvous on the control channel. Second, they transfer one or more packets on the agreed data channel. The two stages saturates under different conditions.

The rendezvous process can generate at most $\frac{1}{e}$ or 0.3679 new agreement during every slot, on average. (We assume contention is successful with probability $p_{succ} = e$.) In the 802.11a scenario, each 1024-byte packet lasts roughly 7 time slots. If there were no queueing, to fully utilize the 12 channels, the rendezvous process must generate $12/7 > 1$ new channel agreements per time slot which is impossible. Therefore, the control channel saturates before the data channels do, as the load increases.

When the control channel saturates at a load of roughly $\frac{1}{e}/\frac{12}{7}$ or 21.46%, the throughput is (0.2146 $\times$ 6 Mbps/channel $\times$12 channels) or 15.45 Mbps. Threfore, at a load under 15.45 Mbps, the delay is very low because both stages are under-utilized. Once the first

stage starts to saturate, the delay increases quickly for the first time. This explains the first jump in the delay at around 15 Mbps.

As the load increases, the number of queues waiting to be serviced in stage 1 remains constant due to the fixed periodic traffic pattern, the average service delay in stage 1 is relatively insensitive to load. Since the total delay is dominated by the stage 1 delay before the second stage saturates, the delay rises very slowly. Finally, when the load approaches the capacity of the data channels, the second stage saturates, and hence the delay increases rapidly again.

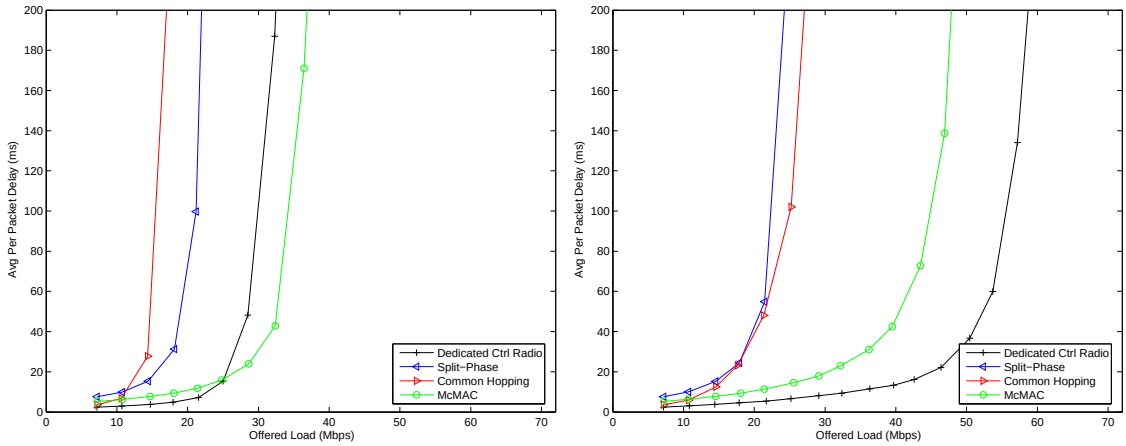
4.6.5 Impact of Limiting Medium Occupancy Time

In this subsection, we evaluate the delay performance using a mixture of long-lived CBR flows and some file transfers. Allowing a device to transmit until the queue becomes empty can help achieve a higher throughput because more data can be transferred per channel agreement. However, the delay and/or jitter experienced by CBR flows might be worse because we do not treat CBR traffic differently from file traffic. We are interested in the effects of limiting the maximum occupancy time on the average delay experienced by CBR flows.

We randomly pair up each device with another and add a CBR connection between them. The degree of communication is therefore 1. The CBR connections have the same data rate, and sum up to 10% of the channel utilization. Then we start random i.i.d. file transfers with geometrically distributed lengths among every pair. The mean file size is 10KB. The arrival rate of the files is adjusted to give a total offered load between 10% (i.e., no file traffic) to 90% (i.e., 80% file traffic). MAC protocols send both types of packets equally in a first-come-first-served order for the same destination.

We ran the simulation experiments with two sets of network parameters. However, due to the similarity between the two, we present only the results pertaining to the 802.11a scenario. Fig. 4.9 shows the delay experienced by CBR traffic under 802.11a with occupancy limits of (a)3.3 ms and (b)10.5 ms respectively. Fig. 4.10 shows the same delay under 802.11b settings with occupancy limits of (a)4.6 ms and (b)15.5 ms respectively. The jitter curves follow the same trend as the delay ones, and are omitted.

In all cases, the increase in the medium occupancy limit reduces the delay for the CBR traffic. Furthermore, the standard deviation of delay (not shown) is almost identical in



(a) (b)

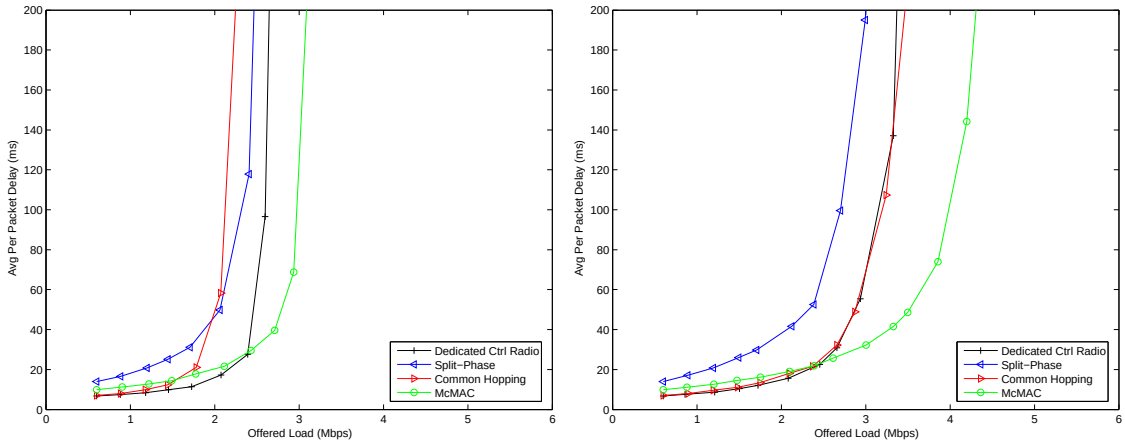
Figure 4.9. The effects of medium occupancy limit on delay v.s. throughput under the 802.11a scenario. Graph (a) shows the case when senders have to relinquish the medium more quickly (3.3ms) than in (b) (10.5ms).

value to the delay. Quite suprisingly, both delay and jitter are lower as one increases the medium occupancy limit. Allowing devices to occupy the medium longer not only benefits the file transfer traffic by increasing throughput, but also the CBR traffic by reducing delay and jitter for all.

Simulation results show that Split Phase does not benefit from a longer medium occupancy limit. As seen in Fig. 4.7 and Fig. 4.8, when the traffic is non-disjoint, the performance of Split Phase is severely impacted because a device can only communicate with a small subset of others during each data phase. Due to the small amount of traffic sent to each destination, a longer occupancy limit is not helpful.

Common Hopping improves tremendously as one increases the occupancy limits indicating that the rendezvous process is indeed a bottleneck. McMAC improves moderately but not drastically because the reduction in rendezvous traffic has less effect on parallel rendezvous protocols because rendezvous is less of a bottleneck.

The throughput of Dedicated Control Channel increases dramatically as the medium occupancy limit increases under the 802.11a settings with 12 channels because the rendezvous is indeed the bottleneck. It improves only a little in the 802.11b case with 3 channels (Fig.4.10) because rendezvous is not the bottleneck.



(a) (b)

Figure 4.10. The effects of medium occupancy limit on delay v.s. throughput under the 802.11b scenario. Graph (a) shows the case when senders have to relinquish the medium more quickly (4.6ms) than in (b) (15.5ms).

4.7 Conclusion

We classified various multi-channel MAC protocols into 4 generalized categories: Dedicated Control Channel, Common Hopping, Split Phase, and McMAC. We then compared them using analysis and simulation. We developed analytical models for the four protocols using Markov chains and simulated them in a time slotted simulator under a variety of operating conditions. The findings are summarized in Tab.4.1. Overall, when channels are numerous and packets are short, control channel congestion occurs in all single rendezvous protocols.

Finally, we found that when MAC protocols do not schedule various packet types (e.g., real-time video and file transfers) differently, allowing a sender to transmit multiple back-to-back packets to the same destination after rendezvous is highly beneficial. The throughput, delay, and jitter of all packets types improve, using any of the 4 protocols.

Parameter Investigated	Dedicated Control Channel	Split Phase	Common Hopping	McMAC
Scalability to Many Channels	Long pkts – Good Short pkts – Limited	Limited	Limited	Good
Sensitivity to Packet Length (Lower is better)	Many channels – High Few channels – Low	High	High	Medium
Sensitivity to Channel Switching Time (Lower is better)	Low	Low	Very High	High
Sensitivity to Receiver Contention (Lower is better)	Low	Medium	High	Medium

Table 4.1. Summary of comparison of four representative protocols under different operating conditions.

Chapter 5

Design of a New Protocol: McMAC

5.1 Protocol Design Rationale

In the previous chapter, we compared the performance of four major approaches to designing multi-channel MAC protocols. When channels are numerous and the amount of data transfer is short after each rendezvous, single rendezvous protocols all suffer from control channel congestion. To get around this problem, consider the following simple protocol called *McMAC without Hopping*:

- **Home Channel:** Each node independently and randomly chooses one of the channels as its home channels as shown in 5.1. When the node is idle, it simply listens on its home channel for any packets belonging to itself.
- **Initialization:** During startup, each node scans through all channels to discover neighbors. Existing nodes beacon periodically on their home channels. The newcomer can either passively listen for beacons or actively send out broadcast probe packets to quickly solicit responses.
- **Receiving:** To receive, a nodes simply listens on its home channel for any incoming packets.
- **Sending:** To send a packet, the sender first determines the home channel of the

	T=1	T=2	T=3	T=4	T=5	T=6	
Channel 1	A, E	A, E	A, E	A, E	A, E	A, E	...
Channel 2	C	C	C	C	C	C	...
Channel 3	B	B	B	B	B	B	...
Channel 4	D, F	D, F	D, F	D, F	D, F	D, F	...

Figure 5.1. McMAC without Hopping. Nodes independently choose their home channels.

receiver. Once determined, the sender tunes its transceiver to the receiver's home channel. It then follows normal CSMA medium contention procedure to send the data packet. See Fig.5.2 for an illustration. If the data packet is successful, the receiver returns an acknowledgement. Otherwise, the sender returns to its home channel and tries again after a random backoff delay.

The advantage of this protocol is that different pairs of nodes can rendezvous on different channels simultaneously, thereby avoiding the bottleneck of a single contention channel. This simple protocol would work well if the active receivers are distributed uniformly over all the channels. However, if busy receivers happen to choose the same channel, that channel would become very congested while others remain idle. Therefore, this simple protocol is insufficient for application scenarios demanding a high throughput.

The basic idea of McMAC is to improve on the simple protocol by requiring each node to use a pseudo-random hopping sequence instead of a constant channel as its *home sequence*. This serves two purposes: load-balancing and robustness. Since the home channel of each node is random, it is very unlikely that busy receivers choose the same home channel all the time. Traffic is therefore spread over every channel. Fig.5.3 shows 4 nodes hopping in a pseudo-random fashion independent of each other. Certain channels might not have any occupants during certain times. The hopping pattern of a node is completely specified by the integer seed used to generate the sequence and the index of the current hop into the sequence.

Another benefit of using a hopping sequence is that an interferer or jammer on a fixed channel will only degrade the performance of McMAC, rather than causing service outage. This is because a receiver changes its channel many times a second following its home sequence. A receiver would only stay on a jammed channel for a small fraction of a second before hopping again. By using a hopping sequence, McMAC avoids interference proactively, rather than changing its home channel reactively after jamming is detected. Without con-

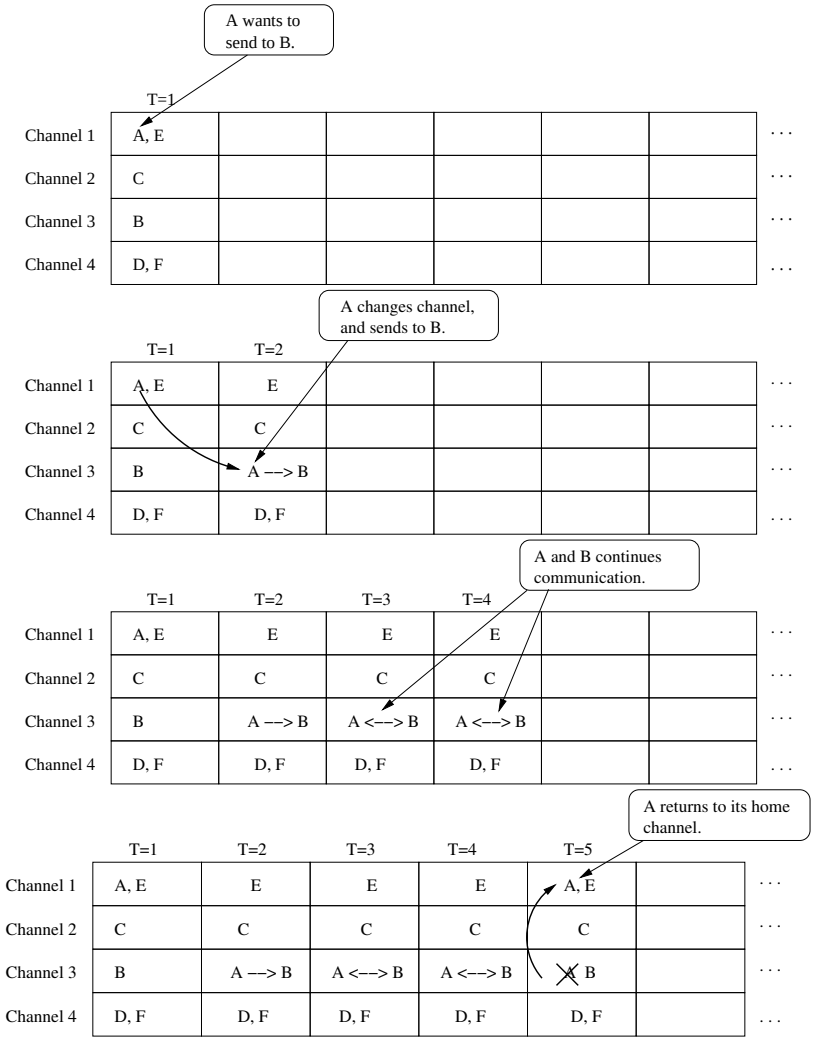


Figure 5.2. Using McMAC without Hopping, node A changes its channel to send data to B.

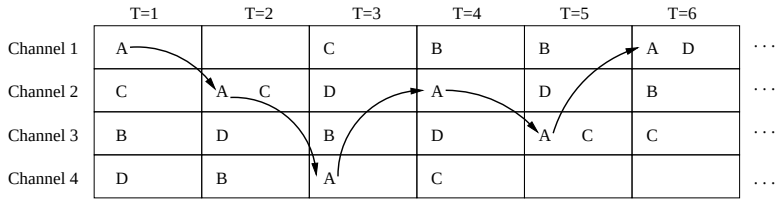


Figure 5.3. McMAC with Hopping. Each node picks a seed and generates an independent hopping sequence to use as their home channel. The hopping sequence of A is 1-2-4-2-3-1-...

stant hopping, upon detecting a jamming signal, a receiver would have to search for a new home channel and then inform all neighbors of its new home channel. The problem is that notifying all neighbors takes significantly longer than simply waiting for the next hop. The disadvantage of using hopping is that neighbors must track the hopping boundaries of each other using a time synchronization mechanism. Fortunately, only pairwise synchronization is necessary.

5.2 Protocol Description

In this section, we describe the details of the McMAC protocol while making only general assumptions of the radio hardware used. Because certain parameters such as the receive-to-transmit turnaround time are hardware dependent, we will list such parameters, and give only a rough range for such parameters. In the next chapter, we will present a specific instance of McMAC that is tailored to our experimental hardware platform. To facilitate the description of the protocol, we divide the description of the protocol into 5 components:

1. **Random Channel Hopping:** Devices hop according to a pseudo-random hopping sequence of their choice. This home sequence is fixed although temporary deviation is allowed.

2. **Discovery:** When a device is turned on, it discovers its 1-hop neighbors and adds them to its neighbor table. Subsequently, it adds neighbors to or removes them from the list when network topology changes.

3. **Synchronization:** Each node estimates the relative speed of the clocks of its neighbors accurately in order to track the hopping boundaries of its neighbors.

4. **Rendezvous:** In order for a node to send a data packet, it must decide when to

change its channel to match its receiver's. It also needs to decide when to send and when to listen for incoming packets. Multiple nodes may decide to send at the same time on the same channel. The access to the medium must be resolved in a fair and efficient manner.

5. **Scheduling:** Each node may simultaneously have data to send to several neighbors. Scheduling is concerned with the order in which a sender visits its receivers which affects the efficiency of the protocol. For example, choosing the neighbor with the most data packets queued may increase throughput.

5.2.1 Random Channel Hopping

Time is divided into Small Slots (T_s) and Big Slots (T_b). Each small slot should be roughly equal to the shortest time it takes for the radio to detect a carrier when using Carrier Sense Multiple Access (CSMA). Typical values should be on the order of $10\mu s$. Each Big Slot consists of an integer multiple of Small Slots. A Big Slot always begin on a Small Slot boundary. Nodes hop to the next channel in its hopping sequence whenever it crosses a Big Slot boundary.

McMAC intentionally allows nodes to keep independent slot boundaries to avoid requiring global synchronization within the same network. Hence, nodes do not hop at the same time in general. Fig.5.4 shows the unaligned slot boundaries of 2 nodes. If nodes were required to hop at the same time (i.e., all nodes must establish a global clock), two problems would arise. First, nodes must re-align their slot boundaries whenever two disjoint networks merge. Changing the hopping boundaries can cause transient confusion among neighbors and packet losses. In the worst case, if network partition and merging occurs frequently enough, it may even create instability as different sets of nodes with different original global clocks try to converge on a new common hopping boundary. Second, as the clocks of various nodes inevitably drift apart, each node must compensate for their drift with respect to the global clock. Additional logic is required at each node to make sure they hop according to the global clock, rather than their local clocks.

The internal structure of each Big Slot is shown in Fig.5.5. Each node hops from one channel to another at the beginning of each slot, so some time is reserved for channel hopping. Next, there is a guard time to make sure that nodes that are not perfectly synchronized can still communicate. During the contention window of a receiver, any sender may try to send a packet to it. Depending on the length of the packet, the packet will

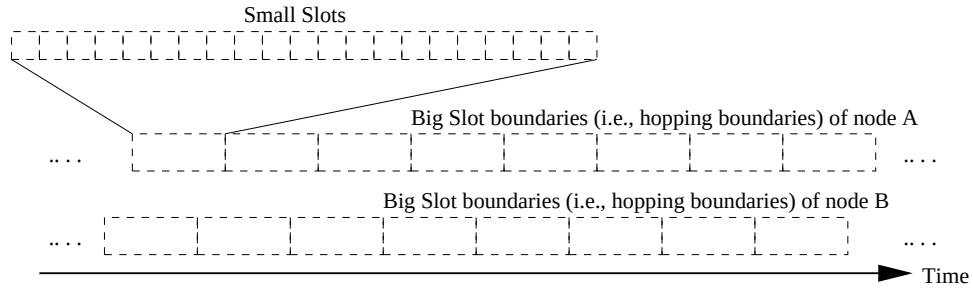


Figure 5.4. Each node keeps independent slot boundaries. Hopping occurs at Big Slot boundaries.

Channel Switch Time	Guard Time	Contention Window	Data Window
Enough for a channel switch	Compensate for Sync. Error	Tx starts sometime within this window	Transmission continues. Data, Ack, and Beacon packets.

Figure 5.5. Each Big Slot is divided into different sections.

continue a different amount into the data window. The data window should be long enough for a maximum size data packet together and the corresponding ack packet.

The hopping sequence of each node is generated using the same pseudo-random generator, with independently chosen seeds. Each device chooses its own seed randomly at startup time, and this seed never changes. Ideally, each device will have a unique MAC address (e.g., Ethernet, 802.11) which can be used directly as the seed. On hardware platforms where the MAC address is longer than the number of bits allowed for a seed, a hash of the MAC address can be used, but the uniqueness of seeds can no longer be guaranteed.

In the rare case where two devices choose the same seed, they might still choose different Big Slot boundaries because they started at different times. In the worst scenario in which their seeds are equal and their Big Slot boundaries coincide, there will be performance degradation only if the two devices happen to be both very active receivers. Furthermore, since the crystal oscillator clocks of different devices inevitably run at slightly different rates, the performance degradation in such rare case is not permanent. It should last the time it takes for 2 clocks to drift 1 Big Slot apart. Assuming a typical drift of 10 parts per million (ppm) and a Big Slot of 4ms, the degradation will last about 400 seconds.

The sequence chosen by a device is known as its home sequence. There are no stringent requirements for the random hopping sequence generated as long as the sequences generated using different seeds behave almost like independent random sequences. The statistical in-

dependence of these sequences are unimportant because they are required for load balancing only, rather than for protocol correctness. In particular, since senders change their channels to meet their receivers when they have packets to send, there is no need for such hopping sequences to coincide frequently. For example, a suitable linear congruential generator by Park and Miller[20] of the form $X(t) = 16807X(t-1) \bmod (2^k - 1)$ can be used to generate the hopping sequence. In this case, $X(0)$ is the seed, and $X(1) \bmod M, X(2) \bmod M, \ldots$ would be used as the hopping sequence where M is the number of channels. As a counterexample, generating different sequences by cyclic-shifting a fixed pseudo-random sequence would not be a suitable for McMAC because different sequences generated this would be highly correlated.

For our purpose, we do not need to use the full period of a pseudo-random hopping sequence. Nodes can let their channel hopping sequence repeat after a fixed number of hops (say k hops). To save computation, a node can pre-compute the first k numbers from the hopping sequence of each of its neighbor. As long as the period k is much larger (e.g., 10 times) than the number of available channels, the hopping sequence will perform satisfactorily for load balancing.

5.2.2 Discovery

Discovery is the process in which nodes learn about their current 1-hop neighbors and their corresponding hopping signatures. The hopping signature of a node is its seed used for hopping sequence generation together with its current local clock value. The hopping signature alone completely specifies the hopping pattern of a node. The clock value is also used for its neighbors to synchronize with it.

As in illustration, suppose a node uses a 32-bit local clock which ticks at 1MHz. Small Slots are $16\mu s$ each. Each Big Slot consists of 256 Small Slots (i.e., 4096 μs). We further assume that the random hopping sequence repeats after 512 hops. Fig.5.6 illustrates the meaning of the different bits in a local clock.

In McMAC, every packet sent includes the hopping signature of the sender, namely the 32-bit current time and the seed of the sender. Since the seed is often the same as the sender's MAC address which is already present in the MAC header, it can be omitted entirely to save space. The space overhead is therefore only a time stamp. The time stamp should be taken when the packet is sent on the medium, after any queueing and processing

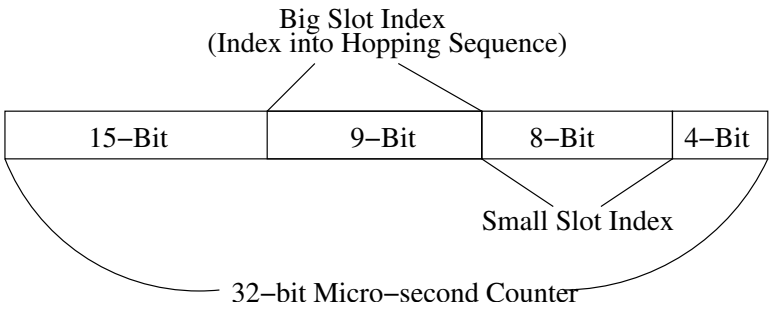


Figure 5.6. In this example, nodes use a 1 MHz clock. Nodes change channel when the last 12 bits of the clock reads 0.

delays in the MAC layer to ensure accuracy. The accuracy of this time stamp is vital to the estimation of a neighbor's clock rate.

Discovery in McMAC relies on two mechanisms: primary and backup. Using the primary mechanism, when a node turns on, it monitors the default discovery channel for beacons from existing nodes for 10 seconds. The default discovery channel can be any of the available channels. Without loss of generality, we assume the default discovery channel is channel 1.

After 10 seconds, the newcomer starts following its home hopping sequence. From then on, the newcomer beacons once every second with a small randomized delay to ensure that it beacons on all channels with equal probability. In addition, it also beacons at least once every 2 seconds when its home sequence lands on the default discovery channel (channel 1). If the beacon on the default discovery channel is delayed for any reason (e.g., medium is busy), a node must retry as soon as possible such that the 2-second rule is maintained.

When a beacon packet is received, if the sender is a new neighbor, the receiver records the hopping signature of the sender consisting of the time stamp and the seed of the sender. Otherwise, the sender's time stamp is used by the synchronization algorithm to update receiver's estimation of the sender's clock drift, such that it can accurately predict the sender's home channel in the future. Fig.5.8 shows the logic used by each node to beacon and discovery its neighbors.

Whenever a new neighbor is discovered, the receiver sends a courtesy HELLO packet after a small randomized delay. This way, the newcomer and the existing node is made aware of each other's existence. If a neighbor has not been heard from for 300 seconds, the neighbor is removed from the neighbor table. The timeout parameter can be tuned to suit specific applications.

Note that it is not necessary to receive a packet of specifically the beacon type in order

to discover a node. Any packet sent in McMAC contains the sender's MAC address (which is also its seed) and the sender's time stamp. These information are sufficient for discovering the neighbor's hopping signature. The beacon packets are there to ensure even otherwise idle nodes beacon periodically to speed up discovery and prevent them from being timed out by their neighbors.

In addition to sending beacons, it is also possible to ask a neighbor for a list of its known neighbors. In such case, the receiver of the list must verify that each node listed is indeed a reachable neighbor of its own. For simplicity, this approach is not included as part of the McMAC protocol specification. Instead, this gossiping process can be considered as an enhancement to McMAC in the future.

Effectiveness of Discovery Mechanisms

Normally, since each node is required to beacon once every 2 seconds on channel 1, listening on channel 1 for 2 seconds is sufficient to discover all existing nodes in a neighborhood. To account for packet loss, each node listens for 10 seconds on this default channel. We expect this primary discovery mechanism to be sufficient in most cases. In the rare event that the primary mechanism fails to discover all neighbors, each node also beacons on all channels randomly every second. For every beacon, there is some chance that a node which did not receive before will now hear it. Eventually, the receiver will have heard at least 1 packet from this sender which is sufficient to discover the node. In Sec.5.3.1, we analyze the delay it takes for a newcomer to discover all existing nodes through passive listening while hopping. In Sec.5.3.2, we study the delay for a newcomer to be discovered by all of its neighbors if only the newcomer sends beacons. In practice, McMAC uses both, together with sending a courtesy HELLO packet to a newly discovered neighbor.

5.2.3 Synchronization

Synchronization is the process in which two neighbors estimate and compensate for the difference in their clock speeds. Fig.5.7 shows the basic idea of the synchronization algorithm. In McMAC, every packet contains the current time of the sender. When received, the receiver time stamps it using its local clock. By plotting receiver's time stamp vs. the sender's time stamp of a sequence of incoming packets, one expects to see a line of slope very close to 1. By observing the difference between the actual line and the ideal line of

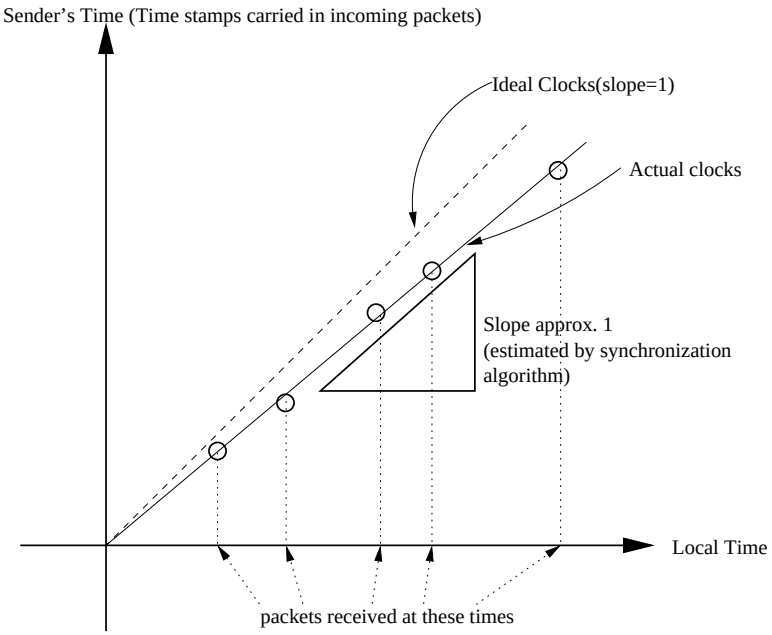


Figure 5.7. The receiver uses a synchronization algorithm to estimate the drift of the sender's clock with respect to its own clock.

slope 1, the receiver can estimate the rate at which the sender's clock drifts apart from its own. Since the sender hops according to the sender's clock, carefully estimating the drift is essential to the receiver's ability to track the sender's hopping sequence and boundary. In order to understand the details of the synchronization mechanism, we must first understand the nature of the clock drift problem. For this reason, we defer the detailed description of the synchronization procedure until the next chapter in Sec.6.4.

All packets (data, acknowledgements, beacons) including those that are overheard can be used for synchronization because every packet sent contains the sender's time stamp. This is very important because when the network traffic load is high, the medium is congested and beacon packets might be lost. If synchronization depends on only beacon packets, neighbors might become poorly synchronized when traffic load is heavy, causing packet losses.

5.2.4 Rendezvous

Rendezvous is the process in which the sender sends a packet to a specific receiver given that the sender has an accurate estimate of the current home channel of the receiver. The sender must also balance the time it spends sending versus receiving. If a node spends all time sending, it will not receive any packets. Using McMAC, rendezvous can happen on

multiple channels simultaneously among several pairs of senders and receivers. The choice of which receiver to send to first is the job of the scheduler which is described in Sec.5.2.5.

At the end of the current Big Slot before a new Big Slot begins, a node decides if it wants to transmit or receive in the upcoming Big Slot. Fig.5.8 shows the flow chart of deciding whether to send or receive at the beginning of each Big Slot. If it has no packets to transmit (i.e., it is idle), it simply tunes its radio to its next home channel according to its hopping sequence and listens for incoming packets until the Big Slot ends. Otherwise, it flips a coin. With probability p_{tx}, it will transmit; with probability $1 - p_{tx}$, it will listen. Ideally, p_{tx} should be adaptive to the traffic load of the network. When the load is high, each node has a smaller time share of the spectrum, so it should spend more time listening. Hence, p_{tx} should be smaller. When medium is not busy, the sender should try to transmit more often to reduce the medium access delay, so p_{tx} should be larger. For simplicity, we used a fixed p_{tx} of 0.2. It turns out that the performance of McMAC in terms of delay and throughput is not very sensitive to the choice of p_{tx} as long as p_{tx} is much less than 1. We expect p_{tx} to be tuned accordingly in each specific system.

Upon deciding to send, the scheduler chooses the appropriate receiver. The sending procedure is illustrated in Fig.5.9. The sender then goes to the current home channel of the receiver at the beginning of its upcoming Big Slot. Within each Big Slot is a number of consecutive Small Slots known as the contention window. The sender picks a random Small Slot within the contention window to begin transmission. During the contention window before its scheduled transmission time, it listens. During the target Small Slot for which the transmission is scheduled to begin, the sender senses for a carrier. If the medium is idle, it begins transmission. Otherwise, it aborts the transmission and return to its own channel. It is possible for the sender to overhear another packet while waiting for its target Small Slot. In such case, the received packet is processed as usual. If the packet lasts beyond the the target Small Slot, the sender will return to its home channel after processing the incoming packet.

After the packet is sent and successfully received, the receiver returns an Ack packet if it has no data to send in reverse. If it has, it can return a Data packet with the Ack bit set instead. Each data packet contains a *More-Data* bit which signals to the receiver whether to expect more packets from this sender. If the sender sets this bit, the receiver should expect at least one more data packet from this sender. The receiver will return 1 Ack packet (or Data + Ack packet) for each data packet received. This back and forth process continues until both sides clears their More-Data bit. McMAC allows packet trains

because the overhead of rendezvous and acquiring the medium is higher in a multi-channel MAC than in a single-channel MAC. To amortize these costs, nodes are allowed to send multiple packets after rendezvous. The procedure of incoming packets and the generation of Ack or Data + Ack packets in response is described in detail in Fig.5.10.

5.2.5 Scheduling

A multi-channel MAC protocol should schedule packets destined for different neighbors carefully to prevent head-of-line blocking and to improve performance. Head-of-line blocking occurs in a multi-channel situation when a node has packets to send to multiple receivers, but the receiver of the first packet in the queue is busy communicating with another node. In principle, the sender can send the second packet in the queue to another receiver using a different channel to avoid head-of-line blocking.

In McMAC, we assume that each sender keeps a separate queue for each neighbor to avoid head-of-line blocking. There are many ways a sender can decide which receiver to send to first when multiple queues are backlogged. We use a simple randomized algorithm (shown in Fig.5.11) to select a receiver.

At the beginning of a new Big Slot, the sender lists all neighbors for which it has packets waiting to be sent. For these destinations, it determines the beginning of their Big Slots starting within its own upcoming Big Slot. (Note: Big Slot boundaries of different nodes do not coincide.) If at least one neighbor happens to have the same home channel as itself during the upcoming Big Slot, one of such neighbors will be chosen as the destination. The rationale for favoring receivers on the same home channel is that when the network is congested, it is likely that a sender can find one receiver with the same home channel. By sending in one's home channel, the sender can count down to the target transmission time (i.e., waiting to acquire the channel) while listening for its own packet.

However, if no receiver shares the same home channel during the next Big Slot, the sender must deviate from its home channel in order to send. First, among the channels for which there is a receiver, one will be selected uniformly at random. Second, a receiver will be chosen among those on that channel with equal probability. This two step procedure ensures that the sender does not favor channels containing many receivers during the next Big Slot to avoid an increased chance of collision.

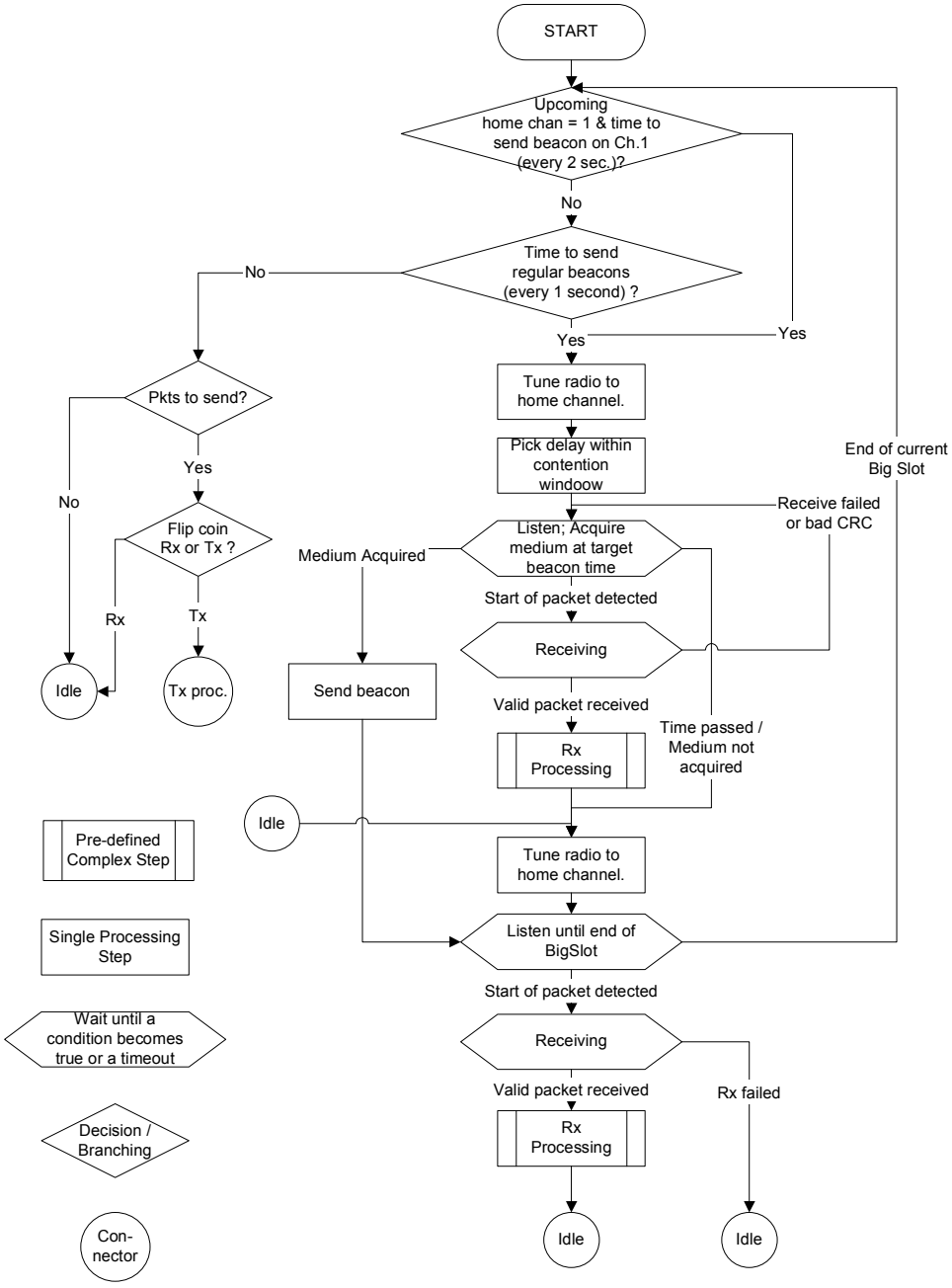


Figure 5.8. This state diagram shows the basic operation of McMAC when a node is not active sending. It only sends beacons and listens for incoming packets.

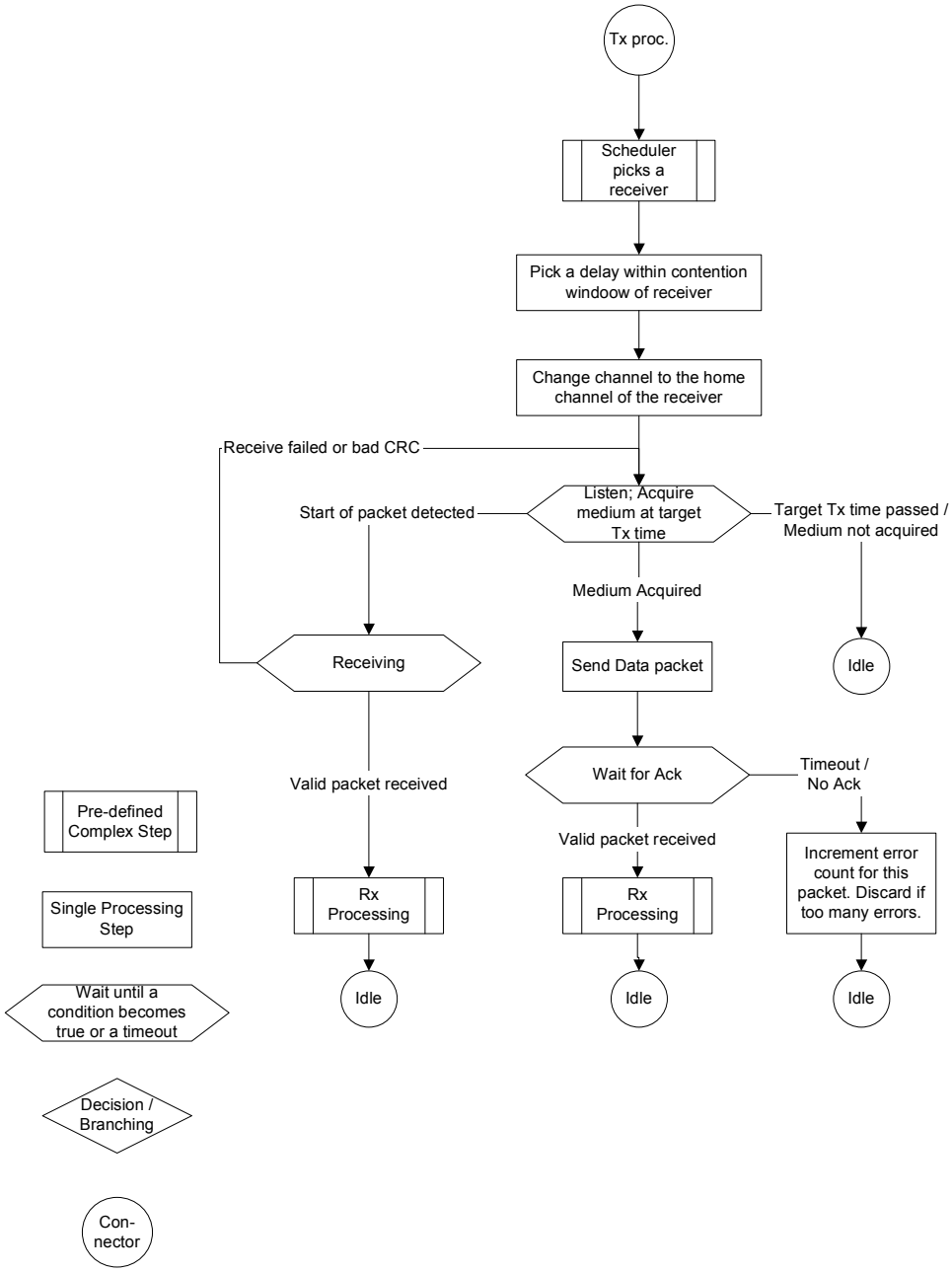


Figure 5.9. This state diagram shows the operation of a node when it is actively sending a data packet.

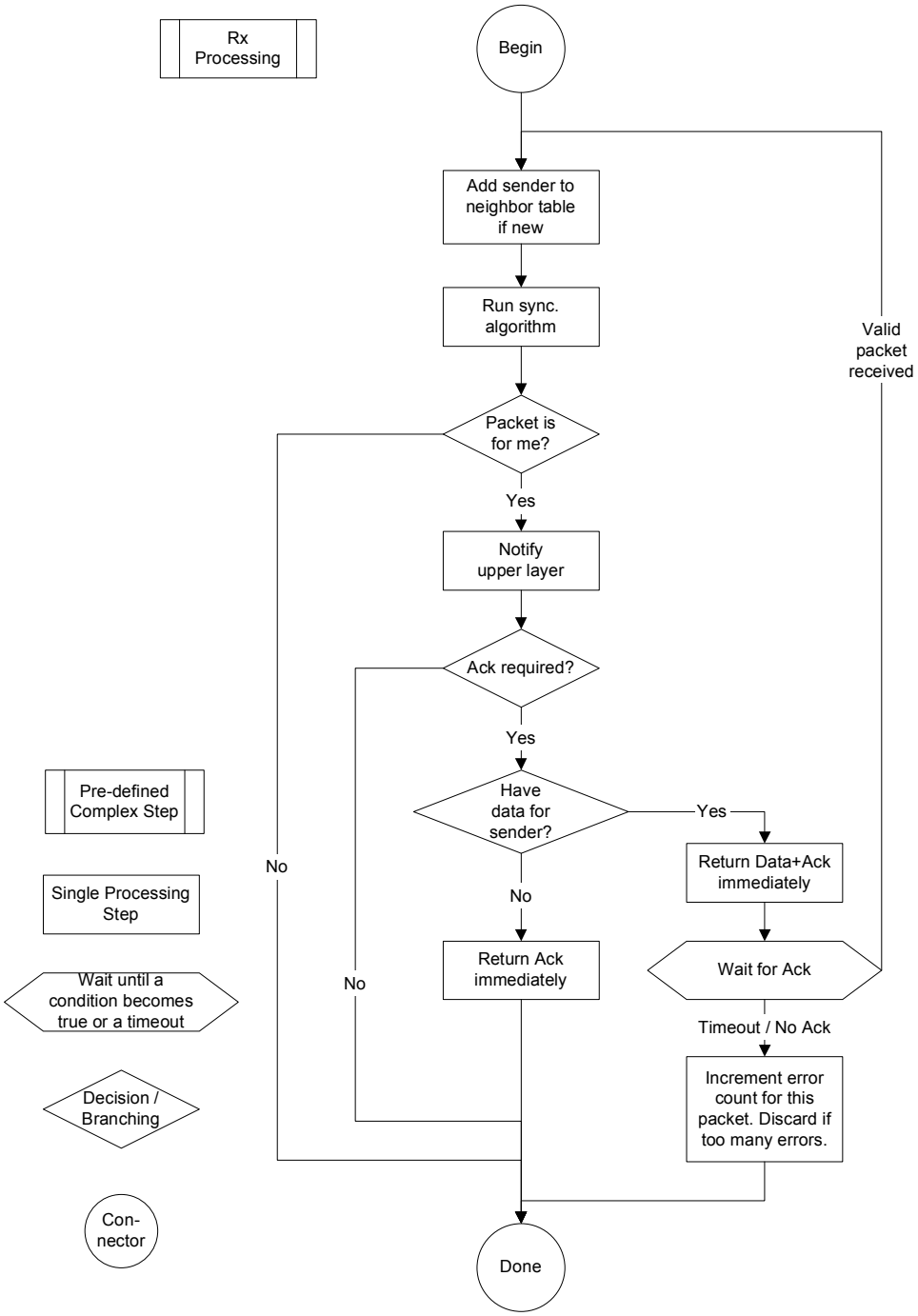


Figure 5.10. This state diagram shows the operation of the receiver

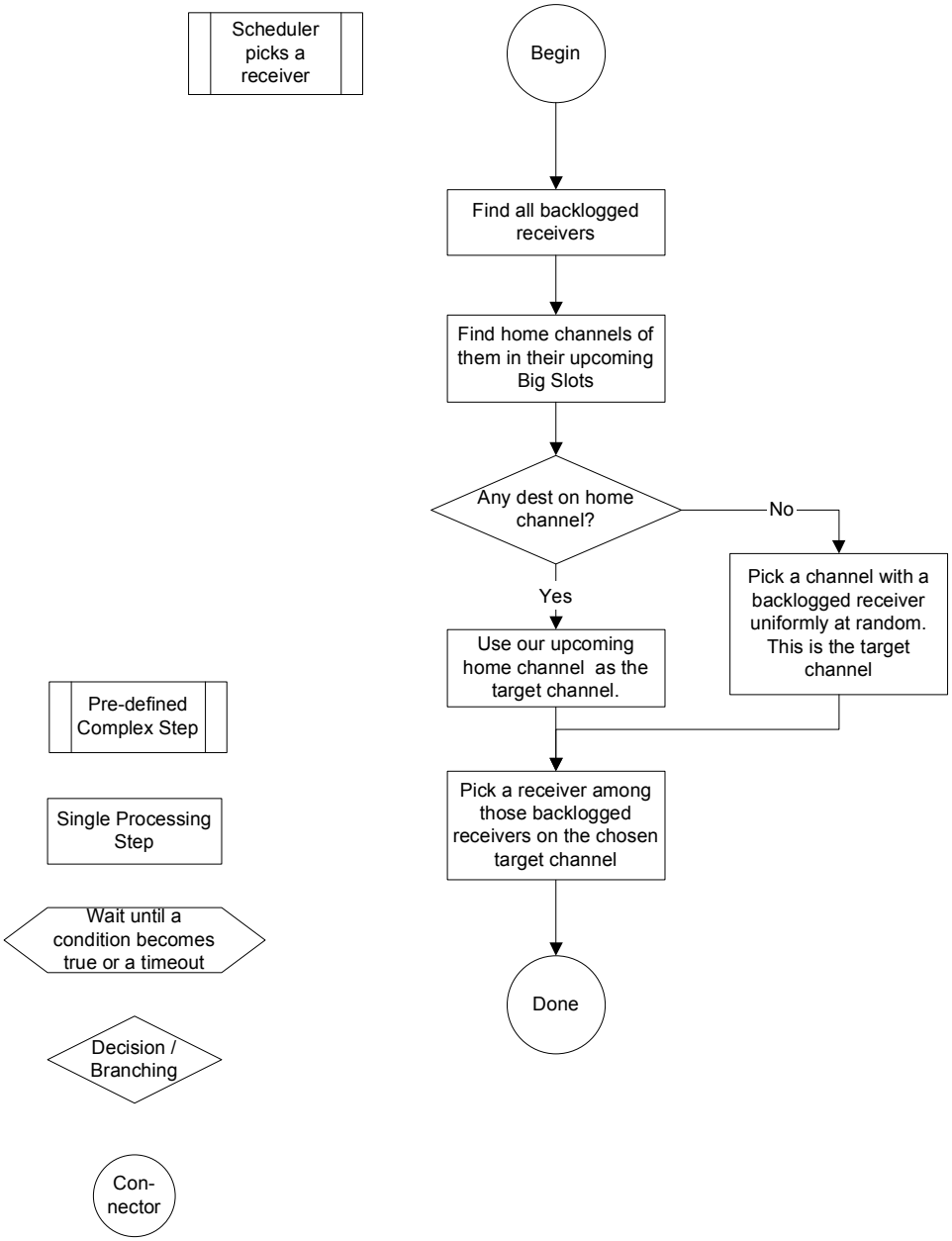


Figure 5.11. This state diagram shows the basic scheduler which favors a receiver who happen to be on the same channel as itself during the upcoming Big Slot. This is done to minimize the time a sender deviates from its home hopping sequence.

5.3 Protocol Evaluation

5.3.1 Delay of Discovering Existing Nodes

In this section, we calculate the expected time it takes for a new node joining a 1-hop multi-channel network to discover all existing neighbors via passive listening. In the next section, we calculate the time for a newcomer to be discovered by its neighbors through active beaconing. This gives us an estimation of how long it takes for all neighbors to be discovered if the primary neighbor discovery method somehow fails. It turns out that the expected time to achieve both is roughly the same. The expected delay scales linearly in the number of channels M and the average beacon intervals $1/r$, but only logarithmic in the number of neighbors N.

First, we calculate the expected time for a new comer to discover all existing neighbors via passive listening. We make the following assumptions:

1. Time is slotted. Each packet takes 1 slot time.
2. There are N existing nodes in the network.
3. During every slot, each of the N nodes chooses one of the M channels as its home channel in uniformly at random in an i.i.d. fashion.
4. Each of the N existing nodes sends a beacon packet on its home channel with probability q independently during each time slot.
5. When a new node X joins the network, it chooses an independent hopping sequence. It listens for beacon packets from existing nodes while hopping.
6. We consider only the 1 hop neighborhood around the new node.
7. If more than 1 existing node beacons on the same channel during the same time slot, no packet will be heard by anyone due to collision.
8. When exactly 1 node beacons on the current home channel of X, it hears the beacon and discovers the sender.

We analyze the expected time for the new node to hear at least 1 packet from each of its neighbors by separating the analysis into two steps. First, we focus on those time slots in which exactly 1 node beacons on the same channel as X. We will use the term *rounds*

to denote such events. One first calculate the expected number of rounds until another existing node is heard. Second, we focus on the time between each round.

Let the time slots during which X successfully receives a beacon packet be $T_1, T_2, \ldots$. Let d_i be the round index during which X discovers i-th new neighbor. Therefore, the times for which X discovers a new neighbor are $T_{d_1}, T_{d_2}, \ldots, T_{d_N}$. For convenience, we define T_{d_0} to be 0. We also define D_i to be $T_{d_i} - T_{d_{i-1}}$.

$$E[T_{d_N}] = E\big[(T_{d_1} - T_{d_0}) + (T_{d_2} - T_{d_1}) + \ldots + (T_{d_N} - T_{d_{N-1}})\big] \tag{5.1}$$
$$= E\big[D_1 + D_2 + \ldots + D_N\big] \tag{5.2}$$
$$= E[D_1] + E[D_2] + \ldots + E[D_N] \tag{5.3}$$

Notice that D_j is the delay to receive a packet from a previously unknown node after having heard from $j-1$ nodes. The number of rounds contained in D_j is a geometrically distributed r.v. with a success probability of $r_j = 1 - \frac{j-1}{N}$. $Pr[D_j \text{ contains } k \text{ rounds}] = (1-r_j)^{k-1} r_j$.

$$E[D_j] = 1/r_j \tag{5.4}$$

$$E[d_N] = 1 + \frac{1}{1-1/N} + \frac{1}{1-2/N} + \ldots + \frac{1}{1-(N-1)/N} \tag{5.5}$$
$$= \frac{N}{N} + \frac{N}{N-1} + \frac{N}{N-2} + \ldots + \frac{N}{1} \tag{5.6}$$
$$= N\big(\frac{1}{N} + \frac{1}{N-1} + \ldots + \frac{1}{1}\big) \tag{5.7}$$
$$= N \sum_{i=1}^{N} \frac{1}{i} \tag{5.8}$$

Since

$$1 + \frac{1}{2} + \cdots + \frac{1}{n} > \int_1^n \frac{1}{x} dx = ln(n) > \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n},$$
$$ln(n) < 1 + \frac{1}{2} + \cdots + \frac{1}{n} < ln(n) + 1.$$

In fact, the Euler's constant γ is defined to be:

$$\gamma = \lim_{n\to\infty} \sum_{k=1}^{n} \left(\frac{1}{k} - ln(n)\right) \approx 0.57721566\ldots$$

Therefore,

$$E[\text{num of rounds till } T_{d_N}] \approx N ln(N) \tag{5.9}$$

During each time slot, the probability that any existing node will beacon in the same slot as the new node is r/M. Since each existing node hops and beacons independently, the probability that exactly 1 of the N existing node beacons on the same channel as the new node is $N\frac{r}{M}\big(1-\frac{r}{M}\big)^{N-1}$. The expected time between each round is $\left(N\frac{r}{M}\big(1-\frac{r}{M}\big)^{N-1}\right)^{-1}$.

Since which node wins a round is independent of the round index,

$$E[T_{d_N}] \approx Nln(N)\left(N\frac{r}{M}\big(1-\frac{r}{M}\big)^{N-1}\right)^{-1} \tag{5.10}$$

$$= ln(N)\left(\frac{r}{M}\big(1-\frac{r}{M}\big)^{N-1}\right)^{-1} \tag{5.11}$$

In general, since there are many channels and r less than 1, $\frac{M}{r}$ is large. As a result:

$$\big(1-\frac{r}{M}\big)^{N-1} = \big((1-\frac{1}{\frac{M}{r}})^{\frac{M}{r}}\big)^{\frac{r}{M}(N-1)} \tag{5.12}$$

$$\approx e^{-\frac{r}{M}(N-1)} \tag{5.13}$$

The average number of nodes trying to beacon is $r \times N$, which must be much smaller than M, or else the channels will be flooded with only beacon messages. When $r \ll \frac{M}{N-1}$, we can approximate $\big(1-\frac{r}{M}\big)^{N-1}$ by 1 according to (5.13). Therefore,

$$E[T_{d_N}] \approx \frac{Mln(N)}{r}$$

.

5.3.2 Delay of Discovering a New Node

In this section, we calculate the expected time for a newcomer to be discovered by its neighbors through active beaconing. The following assumptions are made:

1. Time is slotted. Each packet takes 1 slot time.
2. There are N existing nodes in the network.
3. Each of the N existing nodes hops according to its own independent hopping sequence. Each node listens on one of the M channels in an i.i.d. uniformly-random fashion.
4. When a new node X joins the network, it starts sending beacon packets with probability p during each time slot.
5. We consider only the 1 hop neighborhood of the newly arrived node.

6. A beacon packet sent by the newcomer is heard by all nodes listening on that channel during that time slot without error. Nodes on other channels do not hear this beacon packet.

7. Since only the newcomer is beaconing, there is no collision.

During each time slot, X beacons with probability p. Suppose we only consider those time slots for which X beacons. We label all such slots *beacon rounds*. The number of beacon rounds it takes for each of the existing nodes to discover X are i.i.d. geometric random variables with a success probability of $1/M$ during each round. Let R_i be the number of beacon rounds necessary for existing node i ($1 \leq i \leq N$) to hear the first packet from X. Define $R = max(R_1, R_2, \ldots, R_N)$. During each beacon round, whether an existing node hears the newcomer is independent of whether other existing nodes do.

$$\begin{aligned} Pr[R \leq k] &= Pr[R_1 \leq k \cap R_2 \leq k \cap \cdots \cap R_N \leq k] && (5.14)\\ &= Pr[R_1 \leq k]Pr[R_2 \leq k] \cdots Pr[R_N \leq k] && (5.15)\\ &= \big(Pr[R_1 \leq k]\big)^N && (5.16)\\ &= \big(1 - Pr[R_1 > k]\big)^N && (5.17)\\ &= \big(1 - (1 - \frac{1}{M})^k\big)^N && (5.18) \end{aligned}$$

$$\begin{aligned} Pr[R = k] &= Pr[R \leq k] - Pr[R \leq k-1] && (5.19)\\ &= \big(1 - (1 - \frac{1}{M})^k\big)^N - \big(1 - (1 - \frac{1}{M})^{k-1}\big)^N && (5.20) \end{aligned}$$

The expected number of beacon rounds required for every neighbor to have received a packet from X is:

$$\begin{aligned} E[R] &= \sum_{k=1}^{\infty} kPr[R = k] && (5.21)\\ &= \sum_{k=1}^{\infty} k\bigg(\big(1 - (1 - \frac{1}{M})^k\big)^N - \big(1 - (1 - \frac{1}{M})^{k-1}\big)^N\bigg) && (5.22) \end{aligned}$$

The variance of R is:

$$\begin{aligned} var(r) &= E[R^2] - E[R]^2 && (5.23)\\ &= \sum_{k=1}^{\infty} k^2 Pr[R = k] - \bigg(\sum_{k=1}^{\infty} kPr[R = k]\bigg)^2 && (5.24) \end{aligned}$$

The number of slots between each successive beacon round are i.i.d. geometric random variables with mean $1/p$. Furthermore, these random variables are independent of the number of beacon rounds required (R). Therefore, the expected time T till all neighbors have received a packet from X is:

$$E[T] = E[\text{interval between 2 successive beacon round}]E[R] \tag{5.25}$$

$$= \frac{1}{p}\sum_{k=1}^{\infty} k\left(\left(1-(1-\frac{1}{M})^{k}\right)^{N} - \left(1-(1-\frac{1}{M})^{k-1}\right)^{N}\right) \tag{5.26}$$

Equation 5.26 is precise, but it does not give a good intuition of how discovery time depends on p, M, and N. Next, we derive an approximation for $E[T]$ to get a better intuition.

$$\begin{aligned} E[max\{R_1,\ldots,R_N\}] = {} & E[max\{R_1,\ldots,R_{N-1}\}]+ \\ & E[max\{R_1,\ldots,R_N\} - max\{R_1,\ldots,R_{N-1}\}] \end{aligned} \tag{5.27}$$

$$E[max\{R_1,\ldots,R_N\} - max\{R_1,\ldots,R_{N-1}\}] \tag{5.28}$$

$$= E[(R_N - max\{R_1,\ldots,R_{N-1}\})1\{R_N > max\{R_1,\ldots,R_{N-1}\}\}] \tag{5.29}$$

$$= E[(R_N - max\{R_1,\ldots,R_{N-1}\})|R_N > max\{R_1,\ldots,R_{N-1}\}]\times \tag{5.30}$$

$$Pr[R_N > max\{R_1,\ldots,R_{N-1}\}] \tag{5.31}$$

$$= E[R_N]Pr[R_N > max\{R_1,\ldots,R_{N-1}\}] \tag{5.32}$$

$$= MPr[R_N > max\{R_1,\ldots,R_{N-1}\}] \tag{5.33}$$

Since $R_1 \ldots R_N$ are i.i.d., $Pr[R_N > max\{R_1,\ldots,R_{N-1}\}] \approx 1/N$. This approximation would be exact if R_i's were continuous random variables. We therefore expect this approximation to be valid when M is large.

Hence,

$$\begin{aligned} E[max\{R_1,\ldots,R_N\}] &= E[max\{R_1,\ldots,R_{N-1}\}]+ \\ &\qquad E[max\{R_1,\ldots,R_N\} - max\{R_1,\ldots,R_{N-1}\}] \\ &\approx E[max\{R_1,\ldots,R_{N-1}\}] + M\frac{1}{N} \\ &= M\sum_{i=1}^{N}\frac{1}{i} \\ &\approx Mln(N) \end{aligned} \tag{5.34}$$

As a result, the expected time till every neighbor has heard at least 1 packet from X is:

$$E[T] \approx \frac{Mln(N)}{p} \tag{5.35}$$

Summary

In conclusion, the time it takes for a newcomer X to be heard by all existing node turns out to be the same as the time it takes for X to hear 1 packet from every existing neighbor assuming that they beacon with the same probability p in each slot. This time is equal to $\frac{Mln(N)}{r}$. This delay scales linearly in the number of channels M and the average beacon intervals $1/p$, but only logarithmic in the number of neighbors N.

Chapter 6

Prototype and Experiments

6.1 Practical Challenges

In Ch.4, the performance of the Parallel Rendezvous approach was found to be better than or equal to the other three under most situations. However, this approach, similar to the Common Hopping approach, requires 1-hop neighbor pairs to synchronize and is therefore more complex than the Dedicated Control Channel approach.

Evaluation of MAC protocols through an actual implementation is often very difficult. MAC protocols usually require nodes to following very strict real-time requirements on the order of micro-seconds. In order not to violate such requirements, most MAC protocols are implemented partly in hardware and partly in firmware using a micro-controller. As a result, there are very few platforms that can provide both the flexibility to easily implement new MAC features and the speed to handle packets at link speed. To ensure compatibility of proposed MAC protocols on existing hardware, many new MAC protocols proposed for mesh networks choose to build on top of an existing single-channel MAC protocol.

There are several implementations of mesh network MAC protocols using multiple radios per node. Examples of such research systems include the Multi-radio Unification Protocol (MUP) [2] and Hyacinth [22]. There are also a number of companies [12][13][14] who build mesh network routers using multiple interfaces per node. All of these protocols build on top of 802.11 transceivers to avoid modifying the firmware or hardware of commercial 802.11 chipsets. Therefore, these protocols have to use multiple radios per node in order to use

more than 1 channel. We are not aware of any implementation of multi-channel MAC protocols using a single transceiver per node.

In particular, we are keen on implementing a simple variant of McMAC called McMAC-Light on actual hardware. Through the implementation experience, we hope to achieve these goals:

1. **Prove Feasibility** We hope to prove that the Parallel Rendezvous approach is indeed practical and robust. All multi-channel MAC protocols using 1 radio per node requires some form of synchronization. Protocols requiring time synchronization is often considered fragile. If a node loses synchronization with a neighbor, it will not be able to communicate with it at all, making it impossible to re-synchronize. Fortunately, McMAC require only synchronization between 1-hop neighbors, not global synchronization. Furthermore, even if a node loses synchronization with one of its neighbors, it does not affect its ability to communicate with the others.

2. **Understand Complexity** We also hope to discover potential problems ignored in analysis (for tractability) or neglected in simulation through this implementation exercise. Implementation forces us to consider all the protocol details and weigh this added complexity against the benefits of using a multi-channel MAC protocol. For instance, the level of sophistication of the time synchronization protocol necessary to achieved the required synchronization cannot be easily studied through simulation.

3. **Gain Hardware Design Insights** An implementation will also allow us to identify important features of the radio necessary to achieve good performance. These will serve as guidelines when designing future radio specifically for multi-channel MAC protocols.

6.1.1 Methodology

We start by comparing the pros and cons of several potential hardware platforms in Sec.6.2. Based on the comparison, the most promising platform is chosen. We then benchmark the target hardware platform to better understand the limitations of the platform in terms of throughput, channel switching time, and clock drift. The test results are presented in 6.3. In Sec.6.4, we described the most difficult challenge we face in our implementation: trying to synchronize the neighbors within $200\mu s$ of each other.

Next, we describe the modifications and simplifications made to McMAC, taking into

consideration the limitations of our test platform. In Sec.6.5, we describe the resulting simplified version of McMAC called McMAC-Light which is implemented on our hardware platform. Finally, in Sec.6.6, we evaluate the performance of McMAC-Light and discuss our implementation experience.

6.1.2 Background

Time synchronization is a long-standing problem in distributed systems. In some contexts such as in NTP[18], it refers to the distribution of a reference real-time clock to multiple devices connected by a network. In others, it refers to the establishment of a common clock without any real-time requirements. An example of such protocols is the Flooding Time Synchronization Protocol [16]. In this paper, we are only concerned with pair-wise synchronization between 1-hop neighbors. Therefore, we use the term *synchronization* to refer to the prediction of the current clock of 1-hop neighbors only.

Synchronization among only 1-hop neighbors is considerably simpler than establishing a global clock because:

- Each node estimates their neighbors' clocks individually, so there are worries about whether all estimates converge to a single global clock.
- Queueing and medium access delays happen at only the sender and not intermediate forwarding nodes. This delay is usually known to the sender.
- The propagation delay is bounded by the longest distance between any two possible neighbors.

6.2 Platform Selection

We investigated several potential hardware platforms. First, there is a choice of different controllers. Some platforms such as CaLinx[7] uses an FPGA as a controller while others such as the motes [21] use micro-controllers (MCU). The CaLinx platform is designed specifically for a digital logic design class at UC Berkeley. The Telos mote platform, also from UC Berkeley, is designed for low-power wireless sensor networking. In addition to having different controllers options, we also had a choice of different radio technologies ranging from pure software radio [1], to ASIC 802.11[11][17] and ASIC 802.15.4[3] radios. For our

purpose, we require programmability in all aspects of the MAC layer, but we do not care about the physical layer. Therefore, ASIC radios are a better choice than software radios because they are simpler to program. FPGA-based platforms offers higher performance than micro-controller based ones, but they also require us to program at the Hardware Description Language level which prolongs development time. FPGA-based boards are also more expensive. Finally, for ASIC radios, we had a choice between 802.11[26] and 802.15.4[27] based radios. IEEE 802.15.4 radios are designed for use in wireless sensor networks or lower data rate applications. Therefore, for high throughput systems, 802.11 based radios are more suitable. However, to achieve a high throughput, 802.11 chipsets tend to implement a lot of the MAC features specific to 802.11 in hardware. As a result, the level of control at the MAC level is reduced which is unacceptable because we require tight control of the timing of packets we send and the packet format. The following table summarizes the pros and cons of the different platforms:

	FPGA Software Radio	FPGA + ASIC Radio	Micro-controller + ASIC Radio	Commercial 802.11 Ref. Design
Example Platform	Polarizone SDR Design Bench [1]	CaLinux[7]	Telos[21] SmartStudio[4]	Atheros [11] Maxim [17]
Programming Abstraction Level	HDL	HDL	C or Assembly	C or Assembly
PHY/MAC Programmability	PHY + MAC	MAC	MAC	Parts of MAC
MAC Source Code Availability	Little	Little	Some	Some
Development Time	Long	Long	Short	Short
Unit Cost (USD)	$1000's	$100's	~$100	~$20
Throughput	High	High	Lower	High

Table 6.1. Summary of comparison of different hardware platforms.

We decided to use a micro-controller based platform with an ASIC radio. We found several such platforms with different features. At the end, we chose the Berkeley Telos Mote [29] platform running TinyOS. The advantages of this platform is the availability of extensive free software and relative low cost of hardware which allows us to buy up to tens of nodes. The downside of this platform is that the micro-controller is relative slow, compared with the radio, thereby artificially limiting the throughput of a node.

Telos uses has a 16-bit micro-controller running at 4MHz. It has 10KB of RAM and an

ASIC DSSS radio conforming to the IEEE 802.15.4 standard rated at 250Kbps. Our MAC protocol was designed from scratch and it does not conform to the ZigBee standard. Each Telos mote has a USB connection for downloading program and communicating with a PC. Finally, it has a 32758Hz crystal oscillator clock yielding gives a resolution of 30.5μs.

6.3 Platform Evaluation

In order to guide the precise specification of the protocol, we conducted micro-benchmarks to measure the performance of the Telos mote platform.

6.3.1 Channels

IEEE 802.15.4 standard has 16 channels in the 2.4GHz range. However, due to energy spill over and imperfect filtering, one cannot use all 16 channels simultaneously without interference. Our experiments found that at most every other channels can be used simultaneously without interference. Hence, we use only up to 8 channels in all of our experiments. We measured the time it takes to switch channels directly by issuing a channel switch command and measuring the time it takes. The result is very consistent at 305μs.

6.3.2 Throughput

It takes about 150μs after the packet is assembled in memory before it can be put on air. Most of this time is spent in sending the packet from the micro-controller to the radio chip over a serial bus bit by bit. The default size of packets in TinyOS is 44 bytes, including a preamble, a MAC header, a 28-byte payload, and a CRC checksum field. Each packet therefore lasts about 1.4 ms at the data link speed of 250Kbps. As a result, in the best case, one expects to send no more than 645 packets per second. In practice, one can send much fewer packets per second from a user application in TinyOS even if MAC-level contention backoff is completely disabled. Our preliminary experimental results show that two nodes can only transfer about 150 packets per second.

6.3.3 Clock Drift

Before we can solve the synchronization problem, we need to know the degree of clock drift on the Telos platform. In general, the error in clocks can be attributed to three factors: (i) long-term average deviation of the clock speed from the ideal real-time clock, (ii) short term variation of the clock speed from the long-term average due to variable ambient temperature, and (iii) errors in the time stamping process.

To minimize the third kind of error, time stamps should be done with the help of hardware. We reuse the time-stamping techniques and implementation by Maroti et al. [16] since they use the same TinyOS platform as ours. Using their method, the sender first transfers the outgoing packet into the buffer of the radio chip. The time stamp field of the packet is left empty. Second, the sender signals the radio to start transmitting the packet. The preamble and the start-frame-delimiter is prepended to the packet by the chip automatically. When the radio starts transmitting the start-frame-delimiter, it generates an interrupt to the micro-controller. This interrupt causes the micro-controller to read the crystal oscillator clock and write the time into the empty time stamp field of the packet whose preamble has just been transmitted.

6.3.4 Long Term Drift

We carried out an initial experiment to quantify the degree of clock drift on the Telos platform, we use 19 devices. One device serves as a poller which periodically sends out broadcast packets to solicit time stamps from each of the 17 devices. When each of the 17 devices receives a periodic broadcast, it time stamps the incoming packet using its 32-bit local clock which ticks at 32,768 Hz, and sends the time stamp packet to the base station. The last device acts as the base station to forward all received packets to the attached PC.

Fig.6.1 shows the relative clock drift of each node with respect to the average of the 17 clocks. The x-axis shows the reference time in hours. The average of the 17 clocks is used as the reference time because we do not have a better reference clock. The y-axis shows the difference between the clock reported by a particular node and the average time stamp reported by the 17 nodes, measured in seconds. Due to the definition of the reference clock, about half of the clocks are faster than the reference. The maximum drift between the fastest and the slowest clocks is about 0.6 seconds over a period of 8 hours, which amounts to about a 21 ppm (i.e., 2.1×10^{-5}) maximum drift.

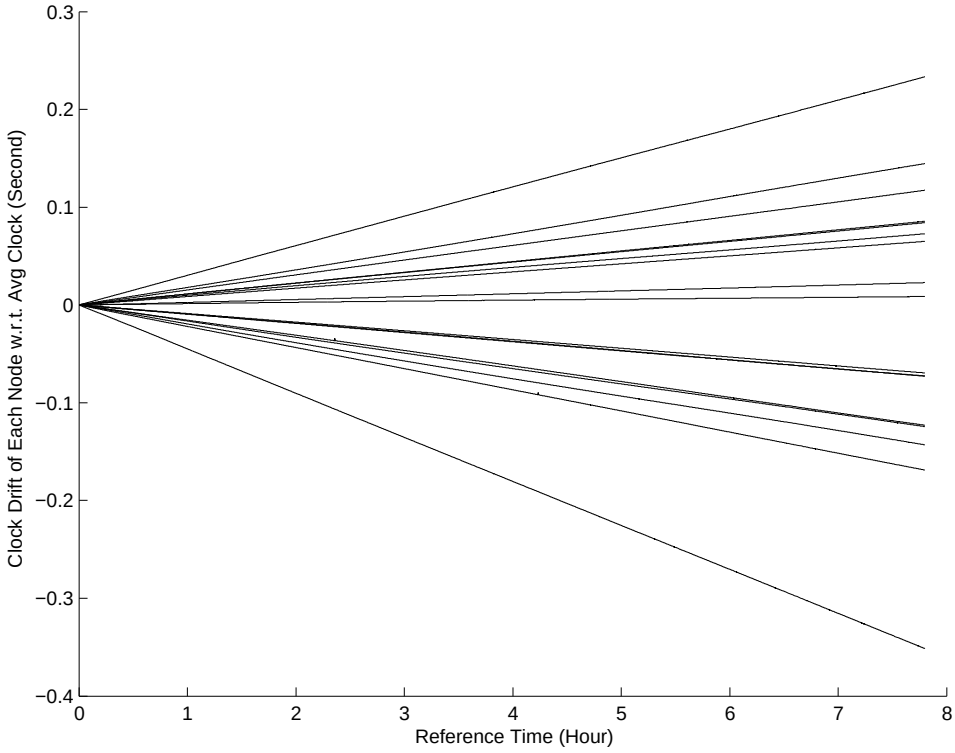


Figure 6.1. Relative clock drift of 17 nodes over 8 hours.

6.3.5 Outlier

There is inherently some small error in the time stamping at the senders and at the receivers. However, during our 8-hour experiment, we observed twice that a receiver erroneously added a huge delay of over 2 ms to an incoming packet. We are unable to pinpoint the source of such error, but we speculate that the micro-controller of the receiver has taken a high-priority interrupt during the reception of the packet, thereby delaying the time stamping interrupt of the incoming packet. For each of these two instances, we concluded that the receiver, rather than the sender, is to be blamed because only 1 out the 17 receivers observed a large error for that packet. Using the average of the receive time reported by all nodes except 14 and 15 as the reference time, Fig.6.3.5 and Fig.6.3.5 show the differences between the reference time and the receive time stamps reported by nodes 14 and 15 respectively. Clearly, nodes 14 and 15 are in sharp disagreement with all others for those two packets in question. We therefore conclude those large errors occurred in the receivers. Throughout other experiments, we also observe an increase in the number of occurrences of such errors when the node is busier, which reinforces our conjecture that the errors are due to delayed interrupts.

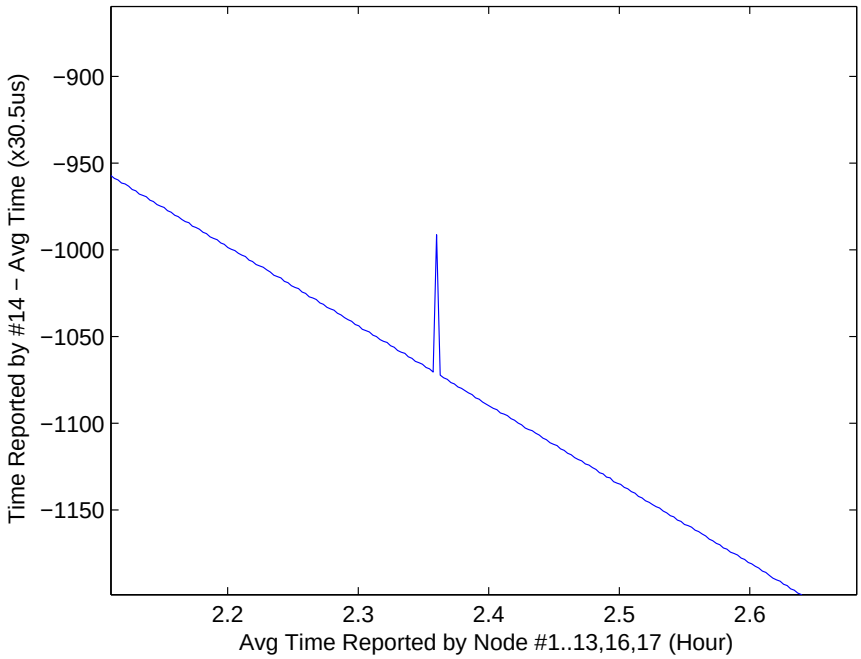


Figure 6.2. Deviation of the clock of node 14 from the average reported by all nodes.

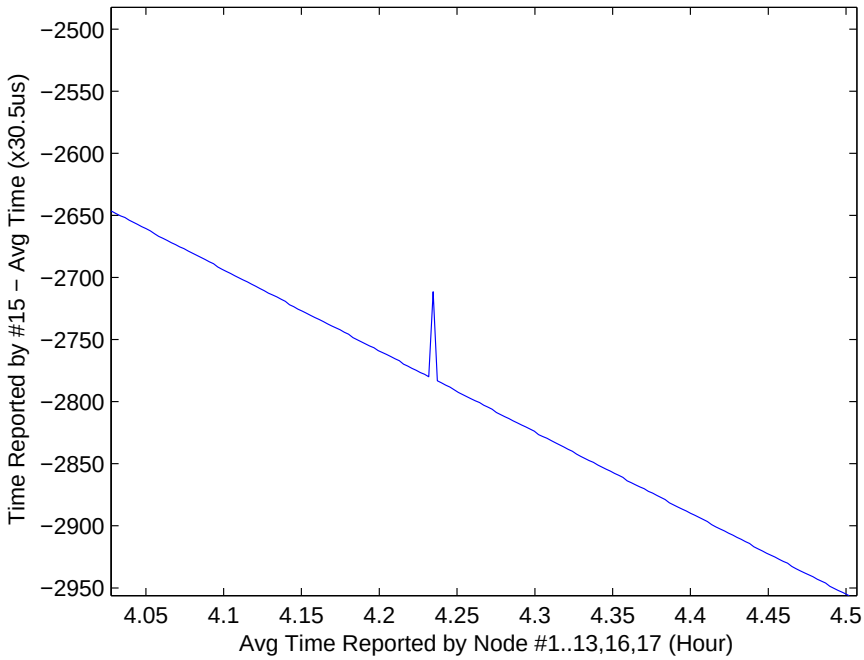


Figure 6.3. Deviation of the clock of node 15 from the average reported by all nodes.

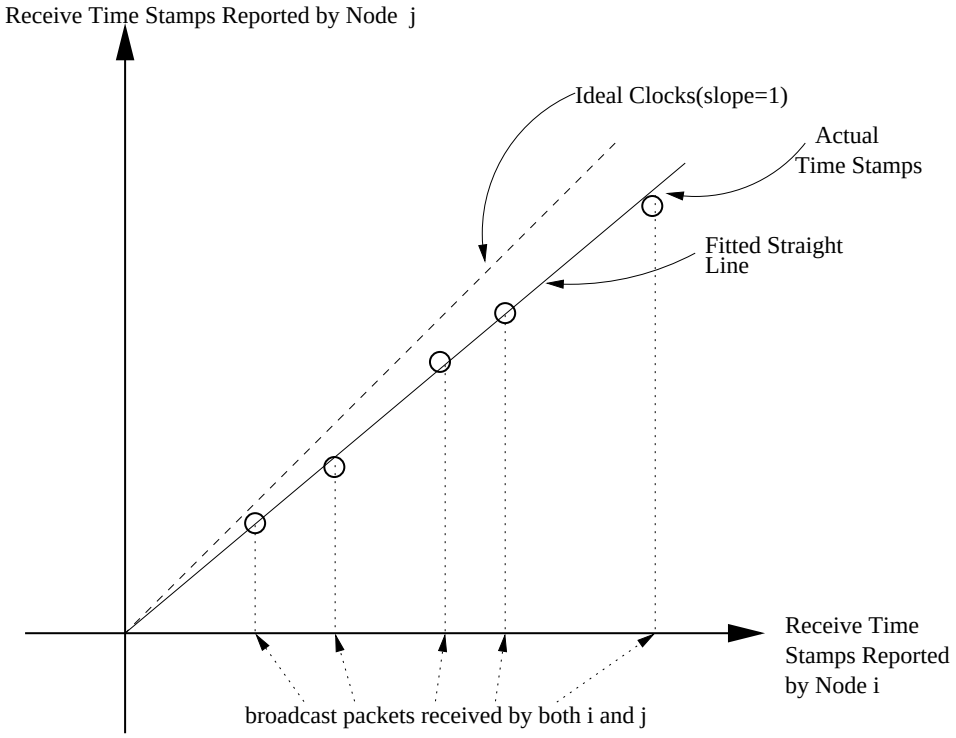


Figure 6.4. For each of the beacon packets received by both i and j, we plot the received time stamps reported by j against those reported by i. Next, we fit a straight line through it. The difference between the straight line and the actual curve is the estimation error.

6.3.6 Short Term Drift Variation

The 17 curves in Fig.6.1 are virtually straight but not exactly so. Therefore, each node has to continuously estimate the drift of their neighbors' clocks using its own imperfect local clock. As a first step towards understanding the clock drift variation over time, we choose arbitrarily 5 pairs among the 17 nodes and plot the receive time stamps reported by node j against those reported by another node i. This would be almost a straight line with a slope very close to 1. Then we fit a straight line through these points using a least squared error (LSE) method. Fig.6.4 shows the idea. Finally, we subtract the straight line from the curve to see the estimation error. Figs. 6.5 6.6 6.7 6.8 6.9 show the residual estimation error for 5 arbitrarily chosen pairs (2,17), (17,11), (11,2), (11,16), and (17,16) respectively.

We observe a few things from these curves:

1. We see some fine noise in the estimation noise. Since the clock resolution is 1 tick (or equivalently 30.5μs), any estimation error below 1 tick is caused by quantization.

2. Certain nodes such as 2, 11, and 17 have clocks that have good short term stability.

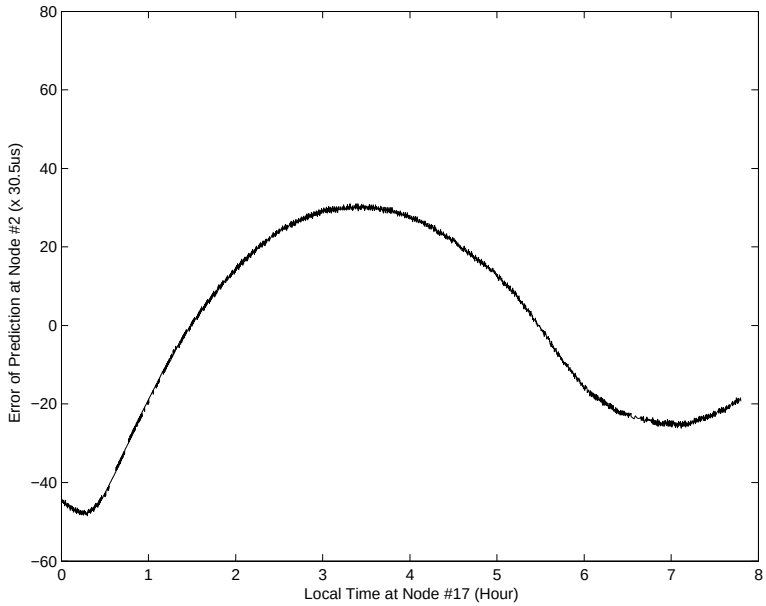


Figure 6.5. Residual Error of Estimation by Regressing Time Stamps of Node 2 on Time Stamps Reported by Node 17.

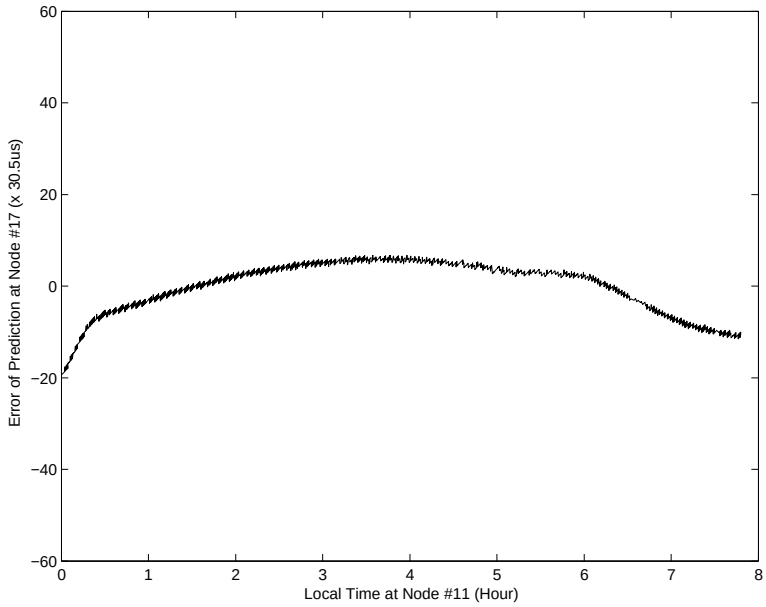


Figure 6.6. Residual Error of Estimation by Regressing Time Stamps of Node 17 on Time Stamps Reported by Node 11.

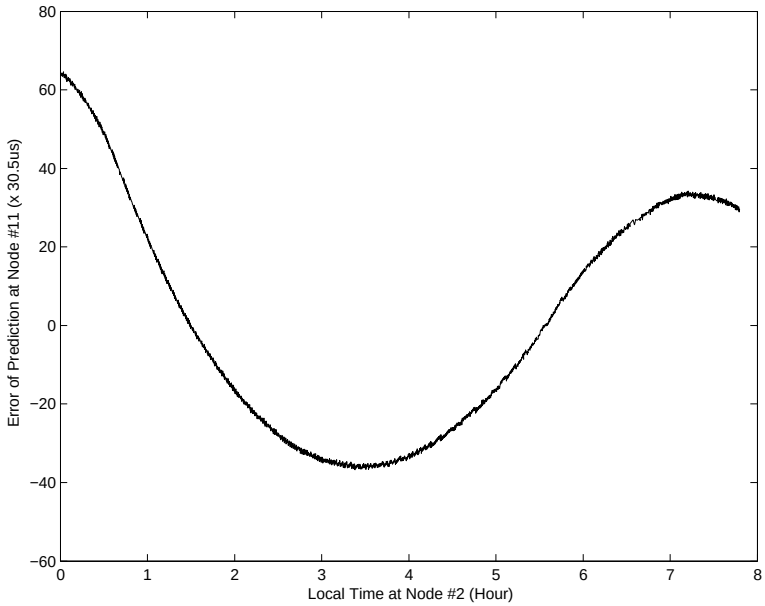


Figure 6.7. Residual Error of Estimation by Regressing Time Stamps of Node 11 on Time Stamps Reported by Node 2.

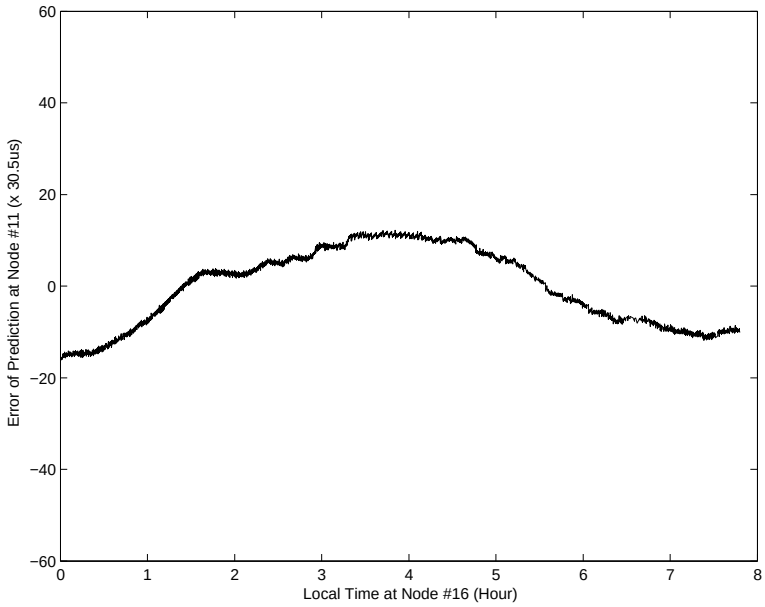


Figure 6.8. Residual Error of Estimation by Regressing Time Stamps of Node 11 on Time Stamps Reported by Node 16.

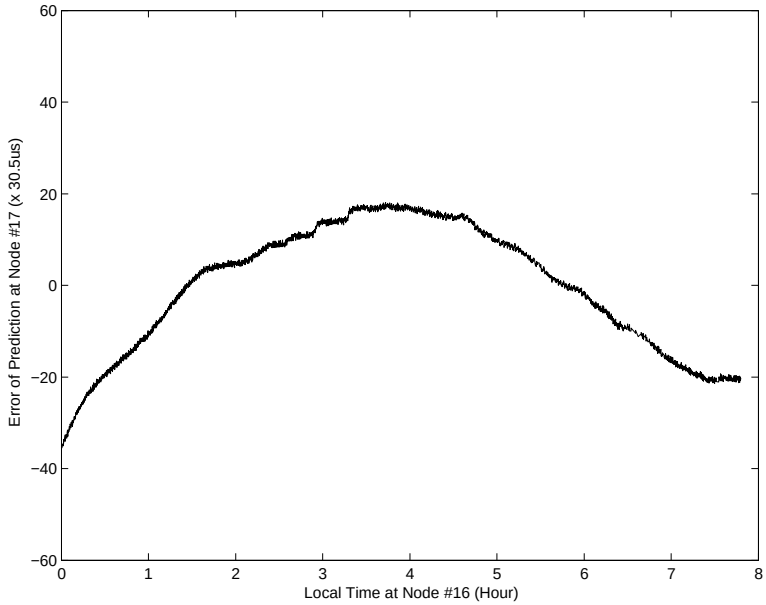


Figure 6.9. Residual Error of Estimation by Regressing Time Stamps of Node 17 on Time Stamps Reported by Node 16.

The general shapes of Figs.6.5 6.6 6.7 are fairly smooth. This is a good news because it means the drift rate of clocks vary slowly for these nodes (at least when they experience about the same temperature). We can correct a large fraction of such estimation error by adaptively estimating the drift rate by examining recent history.

3. Node 16 has poorer short term stability as evident in the presence of more wiggles in Figs.6.86.9.

4. Among the pairs we examined, the curve in Fig.6.7 for node pair (11,2) has the most rapid change in slope. The estimation error changes from +64 ticks at the beginning of the experiment to -16 ticks at the end of 2 hours. This indicates that the relative drift of the two clocks changes about $(64 + 16) \times 30.5\mu s$ over a period of 2 hours, which is equal to 0.34 ppm. In general, this short-term variation of drift between 2 nodes is much smaller than the long-term drift shown earlier in Fig.6.1 which is two orders of magnitudes larger, on the order of 21ppm. Even so, one cannot ignore this short term variation because they can build up errors over time.

6.4 Time Synchronization

In the previous section, we have roughly quantified the clock drift between neighbors. This understanding allows us to investigate different clock synchronization algorithms in this section. To send to receivers, the senders need to know the hopping boundaries accurate to within a small fraction of the Big Slot. Otherwise, the sender might have arrived at the receiver's channel either too early or too late. For the platform we use, the time it takes to switch channel is $305\mu s$ (10 ticks). It means that hopping period should be much longer $305\mu s$. We therefore believe that synchronizing neighbors to within 6 ticks (i.e., $\pm 183.1\mu s$) 95% of the time should be sufficient for our purposes.

To investigate different algorithms for clock synchronization easily, we first collected a set of time stamps from a set of nodes. Then, we simulate different synchronization algorithms under identical packet loss conditions using the collected time stamp traces. Finally, we implement the best algorithm on actual hardware. This greatly simplifies programming because we have the full resources of a PC rather than an embedded micro-controller.

6.4.1 Experimental Setup

In this section, we describe a set of experiments designed to capture time stamps traces for emulating different synchronization algorithms. We set up 16 nodes within the range of each other to each send out beacon packets every 10 seconds. Each beacon packet contains the following information:

1. SrcID (2 bytes): Node ID of the sender.
2. SrcSeqNum (2 bytes): Sequence number of the beacon packet from this sender.
3. SrcClock (4 bytes): 32-bit local time stamp of the sender when the packet is sent.

Upon reception, the receiver sends a log packet containing the received packet and the following information over the USB port to an attached PC.

1. DstID (2 bytes): Node ID of the receiver.
2. DstClock (4 bytes): Local clock of the receiver.

Therefore, when a node beacons, it triggers a total of 15 log packets, one from each of its neighbors. The packets contain sufficient information to emulate any synchronization

algorithm based on nodes periodically broadcasting beacon packets. We can also detect packet losses in our experiment.

6.4.2 MMSE

A standard approach for clock synchronization is by iterative Minimum Mean Squared Error (MMSE) estimator on a recent history of time stamps as in [8]. Let y_i's denote the sender's time stamps of the beacon packets and x_i's be the receiver's time stamps. The index i is the sequence numbers of the beacon packets. We model the relationship between the two as: $y_i = s_{ij}(x_i - x_o) + \epsilon_i$ where ϵ is the error due to the time stamping (on both the sender and the receiver sides). s_{ij} is the ratio of the speeds of j's clock to i's clock. x_o is the value of i's clock when j first starts.

The estimation process goes as follows. The senders send out beacon packets periodically containing the senders' local time stamps (y_i's). The receivers time stamp each beacon packet (x_i's). Based on the recent history of (x_i, y_i) pairs, the receiver predicts the sender's time stamp in the incoming packet. This prediction is called $\hat{y}_i$. The prediction error is $y - \hat{y}_i$. The standard error of estimation is defined to be the square root of the the mean of the $(y - \hat{y}_i)^2$.

Since there are 16 nodes, there are $16 \times 15 \times 2 = 480$ sender/receiver pairs. A synchronization algorithm being tested is run between every pair, yielding 480 standard error figures. We use the median and the maximum of these 480 numbers as the figure of merit for different synchronization algorithms.

It is reasonable to try the MMSE estimator using a variable number of recent send/receive time stamp pairs. If the errors are normally distributed, the MMSE estimatator is the same as the maximum likelihood estimator(MLE), which should perform very well.

The devices such as the motes have very limited memory and processing power, it might not be feasible to remember a large number of previous sample points for regression. However, for now, we ignore the implementation constraints and assume that devices are not resource constrained. We want to know if MMSE performs well enough for our application, given that clock drift rates are not constant, and that errors are not necessarily Gaussian. For example, as we saw in Sec.6.3.5 some time stamp errors are clearly outliers probably caused by delayed interrupts.

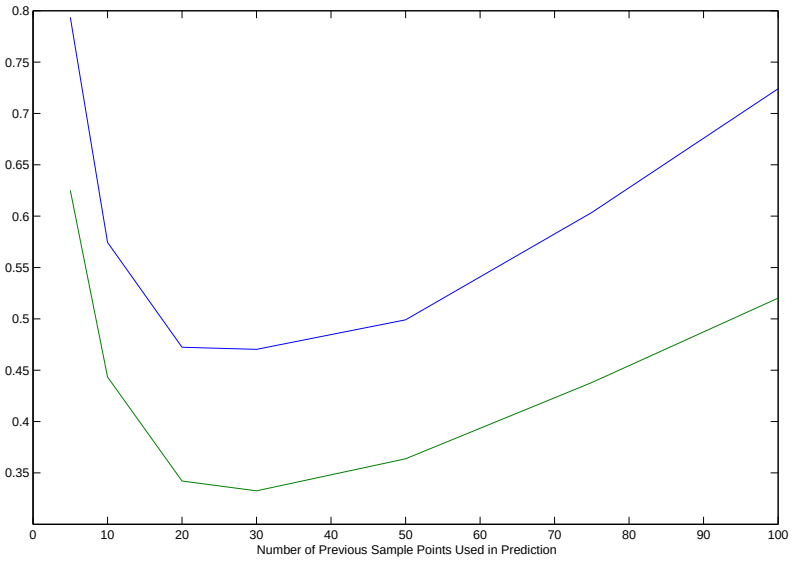


Figure 6.10. Maximum and Median Standard Error of Estimation among All Node Pairs vs. History Length. Data points are 10 seconds apart.

In this section, we use the most recent k pairs of local (receiver's) and remote (sender's) time stamps to feed into a standard MMSE estimator. By varying k, we attempt to find the optimal number of points that should be used to achieve the highest accuracy. If the estimator uses too few points, the error in the time stamp samples would dominate and reduce accuracy. However, if too many points are used, the estimator cannot quickly track changes in short-term clock drift variation.

Fig.6.10 shows the maximum (upper curve) and the median (lower curve) estimation errors as the number of data points used in regression varies from 5 to 100. As expected, the errors first drops but eventually rises again.

The optimal occurs around 30 data points which corresponds to 300 seconds in real-time since the beacons are 10 seconds apart unless there is packet loss. Packet loss ratio between pairs ranges from 0% to 4% with most being below 1%. Therefore, the optimal synchronization algorithm based on MMSE should use about 5 minutes of history.

6.4.3 Recursive Estimation

In the previous section, we assumed that the nodes can remember as many sample points as necessary. However, typical low-cost devices have only limited memory. In particular,

Telos motes have only up to 10KB of memory to be shared by all applications and the operating system. Each time stamp pair takes 8 bytes. Assuming a beacon interval of 10 seconds, to achieve the optimal estimation error requires keeping 30 points or 240 bytes per neighbor. On a different hardware platform, perhaps even more history needs to be remembered. This severely limits the number of neighbors possible.

Therefore, it is preferable to use a recursive algorithm that keeps a smaller constant amount of memory per neighbor and allows an incremental update when a new time stamp pair is observed. Next, we will investigate a recursive estimation algorithm which gives earlier points geometrically lesser weight depending on age instead of equal weights.

Using the same notation as before, x_i and y_i represent the receiver's and sender's time stamps. $\hat{y}_i$ denotes the receiver's estimation of the sender's time stamp. We restrict ourselves to only linear estimators of the form:

$$\hat{y}_i = b_o + b_1 x_i$$

We define the objective of the regression to be minimizing the residual sum of squares as defined below:

$$RSS = \sum_{i=1}^{n} \gamma^{n-i}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \gamma^{n-i}(y_i - b_o - b_1 x_i)^2$$

Setting $\frac{\partial RSS}{b_o} = 0$, we have:

$$\frac{\partial RSS}{b_o} = -2\sum_{i=1}^{n} \gamma^{n-i}(y_i - b_o - b_1 x_i) = 0$$

Equivalently,

$$\hat{\beta}_o \sum_{i=1}^{n} \gamma^{n-i} + \hat{\beta}_1 \sum_{i=1}^{n} \gamma^{n-i} x_i = \sum_{i=1}^{n} \gamma^{n-i} y_i \tag{6.1}$$

Similarly, by setting $\frac{\partial RSS}{b_1} = 0$, we have:

$$-2\sum_{i=1}^{n} \gamma^{n-i} x_i (y_i - \hat{\beta}_o - \hat{\beta}_1 x_i) = 0$$

Therefore,

$$\hat{\beta}_o \sum_{i=1}^{n} \gamma^{n-i} x_i + \hat{\beta}_1 \sum_{i=1}^{n} \gamma^{n-i} x_i^2 = \sum_{i=1}^{n} \gamma^{n-i} x_i y_i \tag{6.2}$$

Solving and directly would yield the desired β_o and β_1, but it would require remembering all x_i's and y_i's as before.

To allow recursive computation of β_o and β_1, we first define the followings:

$$\sigma_{1,n} = \sum_{i=1}^{n} \gamma^{n-i}$$

$$\sigma_{x,n} = \sum_{i=1}^{n} \gamma^{n-i} x_i$$

$$\sigma_{y,n} = \sum_{i=1}^{n} \gamma^{n-i} y_i$$

$$\sigma_{xy,n} = \sum_{i=1}^{n} \gamma^{n-i} x_i y_i$$

$$\sigma_{x^2,n} = \sum_{i=1}^{n} \gamma^{n-i} x_i^2$$

The immediate consequence of the definition is that $\sigma_{1,n}, \sigma_{x,n}, \sigma_{y,n}, \sigma_{x^2,n}$, and $\sigma_{xy,n}$ can be calculated recursively. For example,

$$\sigma_{x,n} = \sum_{i=1}^{n} \gamma^{n-i} x_i = \left(\sum_{i=1}^{n-1} \gamma^{(n-1)-i} x_i \right) \gamma + \gamma^{n-n} x_n = \sigma_{x,n-1}\gamma + x_n \tag{6.3}$$

.

Similarly,

$$\sigma_{1,n} = \gamma \sigma_{1,n-1} + 1 \tag{6.4}$$

$$\sigma_{y,n} = \gamma \sigma_{y,n-1} + y_n \tag{6.5}$$

$$\sigma_{x^2,n} = \gamma \sigma_{x^2,n-1} + x_n^2 \tag{6.6}$$

$$\sigma_{xy,n} = \gamma \sigma_{xy,n-1} + x_n y_n \tag{6.7}$$

By now, we can incrementally update $\sigma_{1,n}$, $\sigma_{x,n}$, $\sigma_{y,n}$, $\sigma_{x^2,n}$, and $\sigma_{xy,n}$ and hence there is no need to keep any time stamps after they have been used to update these 5 variables. The amount of storage needed per neighbor is reduced from the original 30 time stamp pairs (i.e., 60 numbers) down to 5 numbers. Furthermore, as n increases, $\sigma_{1,n} = \frac{1-\gamma^n}{1-\gamma}$ approaches $\frac{1}{1-\gamma}$ which is a constant requiring no updates.

The normal equations (6.4.3) and (6.4.3) can be rewritten as:

$$\hat{\beta}_o \sigma_{1,n} + \hat{\beta}_1 \sigma_{x,n} = \sigma_{y,n} \tag{6.8}$$

$$\hat{\beta}_o \sigma_{x,n} + \hat{\beta}_1 \sigma_{x^2,n} = \sigma_{xy,n} \tag{6.9}$$

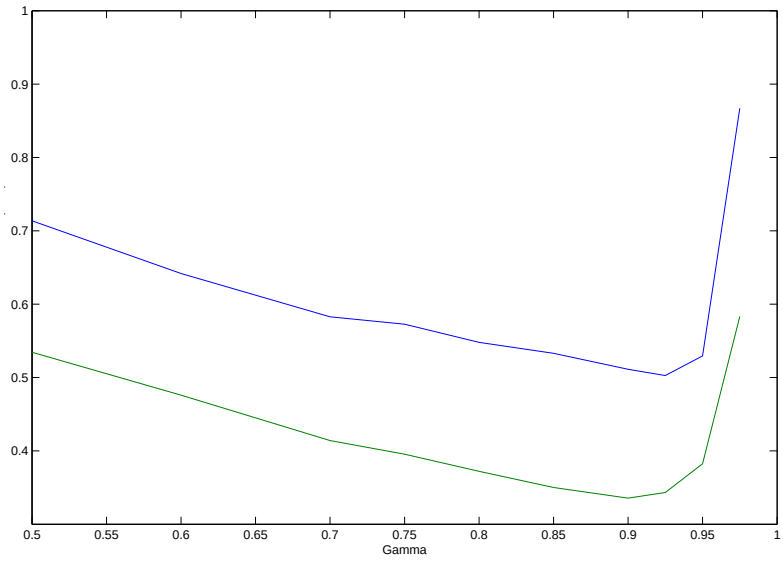


Figure 6.11. Maximum and Median Standard Error of Estimation among All Node Pairs vs. γ. A larger γ gives more weight to earlier points.

Solving eq:normal0a and eq:normal0a, we have:

$$\hat{\beta}_o = \frac{\sigma_{x,n}\sigma_{xy,n} - \sigma_{x^2,n}\sigma_{y,n}}{\sigma_{x,n}^2 - \sigma_{x^2,n}\sigma_{1,n}} \tag{6.10}$$

$$\hat{\beta}_1 = \frac{\sigma_{x,n}\sigma_{y,n} - \sigma_{1,n}\sigma_{xy,n}}{\sigma_{x,n}^2 - \sigma_{x^2,n}\sigma_{1,n}} \tag{6.11}$$

Using iterative MMSE, one design parameter is the number of points to remember for the estimation. In recursive estimation, the forgetting factor γ is important. Fig.6.11 shows the effect of using different values of γ in the estimation accuracy. The x-axis shows the value of γ used, while the y-axis shows the median and maximum standard error of estimation among all pairs. In general, the performance of recursive estimation is not as good as iterative MMSE. However, in practice, both algorithms produces an estimate with a standard error that is less than 1 clock tick. Any error below 1 clock tick is completely overwhelmed by the quantization error. Therefore, there is no practical difference in terms of performance. Also notice that as γ increases the estimation error first reduces slowly, reaches an optimum around 0.925, and then increases very quickly. Therefore, it is safer to err on the side of a smaller γ. It is much more detrimental to forget too slowly than to forget too quickly.

6.4.4 Recursive Estimation without Overflow

The solution presented in equations (6.10) and (6.11) are not suitable for direct implementation because of overflows. As n increases, $\sigma_{x,n}$, $\sigma_{y,n}$, $\sigma_{x^2,n}$, and $\sigma_{xy,n}$ all grow without bound and will eventually overflow.

Take $\sigma_{x,n} = \sum_{i=1}^{n} \gamma^{n-i} x_i$ as an example. First the wrapping of x_i needs to be addressed. Second, as time goes on, the local clock of the receiver x_i increases. The sum $\sigma_{x,n}$ is at least as big as x_i because the weight γ^{n-i} is equal to 1 for the most recent sample. Eventually, the variable containing $\sigma_{x,n}$ overflows.

An important observation is that only the relative values of x_i and y_i are important. Adding a constant offset to the clocks of the senders and receivers does not change the essence of the regression. The motivation for adding an offset to x_i is such that by carefully choosing the offset, we can make the most recent samples of x_i to be close to 0, and earlier samples take on large negative values. In the sum, earlier sample points have an geometrically decreasing weight which quickly approaches zero, therefore the sum $\sigma_{x,n} = \sum_{i=1}^{n} \gamma^{n-i} x_i$ becomes bounded.

To illustrate, suppose we add an offset of $-k$ and $-c$ to x_i and y_i respectively. This amounts to a change of variables from x_i and y_i to $z_i = x_i - k$ and $w_i = y_i - c$. We want to compute $\sigma_{z,n}$, $\sigma_{z^2,n}$, $\sigma_{w,n}$, and $\sigma_{zw,n}$ using $\sigma_{x,n}$, $\sigma_{y,n}$, $\sigma_{x^2,n}$, and $\sigma_{xy,n}$ without redoing the recursive computation from scratch.

$$\sigma_{z,n} = \sum_{i=1}^{n} \gamma^{n-i} z_i = \sum_{i=1}^{n} \gamma^{n-i}(x_i - k) = \sum_{i=1}^{n} \gamma^{n-i} x_i - \sum_{i=1}^{n} \gamma^{n-i} k = \sigma_{x,n} - k\sigma_{1,n}$$

Similarly,

$$\sigma_{z^2,n} = \sum_{i=1}^{n} \gamma^{n-i} z_i^2 = \sum_{i=1}^{n} \gamma^{n-i}(x_i - k)^2 = \sigma_{x^2,n} - 2k\sigma_{x,n} + k^2\sigma_{1,n}$$

$$\sigma_{w,n} = \sum_{i=1}^{n} \gamma^{n-i} w_i = \sum_{i=1}^{n} \gamma^{n-i}(y_i - c) = \sigma_{y,n} - c\sigma_{1,n}$$

$$\sigma_{zw,n} = \sum_{i=1}^{n} \gamma^{n-i}(x_i - k)(y_i - c) = \sigma_{xy,n} - k\sigma_{y,n} - c\sigma_{x,n} + kc\sigma_{1,n}$$

To further simplify, we choose $k = \sigma_{x,n}/\sigma_{1,n}$ and $c = \sigma_{y,n}/\sigma_{1,n}$.

Therefore,

$$\sigma_{z,n} = 0 \tag{6.12}$$

$$\sigma_{z^2,n} = \sigma_{x^2,n} - \frac{\sigma_{x,n}^2}{\sigma_{1,n}} \tag{6.13}$$

$$\sigma_{w,n} = 0 \tag{6.14}$$

$$\sigma_{zw,n} = \sigma_{xy,n} - \frac{\sigma_{x,n}\sigma_{y,n}}{\sigma_{1,n}} \tag{6.15}$$

After the change of variables, we can calculate $\hat{z}_i$ and get $\hat{y}_i$ by adding c to it. The estimation is exactly the same as if we had never changed the variables. In addition, if we changed our variables twice, first by an amount of (k_1, c_1), then again by (k_2, c_2), the result is the same as changing the variables by an amount of $(k_1 + k_2, c1 + c_2)$. The beauty of this is that if we change variable after each iteration by making $\sigma_{w,n}$ and $\sigma_{z,n}$ both zero and then renaming z as x and w as y, we can avoid overflowing $\sigma_{x,n}$, $\sigma_{y,n}$, $\sigma_{x^2,n}$, and $\sigma_{xy,n}$ completely. Algorithm.1 shows the complete recursive algorithm for updating the sums and calculating $\hat{y}_i$.

6.4.5 Computational Efficiency

So far we have ignored the problem of computational efficiency. Since the time stamps are 32-bit long, we have carried out all arithmetic in 64-bit double precision floating point numbers to prevent loss of precision. However, many low cost platforms including our Telos motes do not support floating point numbers in hardware. Worse, the GNU C compiler we use supports only single-precision floating point operations for our motes. If we use only single-precision floating point numbers to implement Algorithm.1, the estimation error would become unacceptably large.

As a result, we can either port a software double precision arithmetic library to our platform, or we can try to carefully craft our calculation to use only single precision floating point numbers. We decided to go with the latter solution because even if we can do double-precision calculation in software, it will still be very slow.

IEEE single precision floating point numbers have 23 bits of mantissa, 8 bits of exponent, and 1 bit for the sign. To minimize any loss of precision, we have to ensure that the

```
/* Initialization */
γ = 0.9        /* Tunable parameter */
READ(x1,y1)  /* From first beacon packet */
i = 1 /* num of beacons received so far */
k = x1
c = y1
σ1 = 1
σx = σy = σx² = σxy = 0

while (True)
    READ(xi,yi)  /* From incoming beacon */
    i = i + 1
    x' = xi − k
    y' = yi − c
    if (i ≥ 3) /* Need 2 points before we can predict */
        ŷi = β̂1 x' + c /* Estimated send time */
        εi = y − ŷi /* Estimation error */
    endif

    /* Update sums */
    σ1 = γσ1 + 1
    σx = x'
    σy = y'
    σx² = γσx² + x'²
    σxy = γσxy + x'y'

    /* Change of variable */
    σz² = σx² − σx²/σ1
    σzw = σxy − σxσy/σ1
    k = k + σx/σ1
    c = c + σy/σ1
    β̂1 = σzw/σz²

    /* Renaming variable */
    σx² = σz²
    σxy = σzw

end while
```

Algorithm 1: Recursive estimation algorithm without overflow.

intermediate numbers in the calculation do not require more than 23 bits of precision. Since each time stamp is already 32-bits, it presents a difficult challenge.

In order to overcome this, we rely on several important observation. First, y_i increases at 32768 ticks/second, but $y_i - x_i$ grows at the relative drift rate between 2 neighbors' clocks which was measured to be no more than 1 tick per second in Sec.6.3.4. Instead of regressing the sender's time stamp y_i on the receiver's time stamp x_i, we can regress $y_i - x_i$ on x_i. Algorithm.1 can be readily reused by feeding it $(y_i - x_i, x_i)$ instead of (y_i, x_i). The only other change is that the estimated sender's time stamp is now $\hat{y}_i + x_i$ instead of $\hat{y}_i$. By using $y_i - x_i$, the magnitudes of σ_y, σ_w, σ_{xy}, and σ_{zw} are much smaller.

Unfortunately, σ_{x^2} does not benefit from this optimization. Next, we turn our focus to reducing x_i. Notice that $y - x$ grows in magnitude at less than 1 tick (i.e.,30.5μs) every 1 second. Assuming that beacons are 10 seconds apart, $y - x$ would change by at most 10. The lower order bits of x is therefore very important in the calculation of $y - x$. However, once we have calculated $y - x$, during the regression of $y - x$ on x, the lower order bits are not important anymore. This is because x grows by 32768 ticks every second. Through experiments, we discovered that we can safely ignore the last 12 bits of receive time stamps after calculating $y - x$.

In addition, the most significant bits turns out to be also unimportant. As an example, suppose that neighbors are timed out after 2 minutes. In 2 minutes, only the lower order 21 bits may change. Therefore, one can safely ignore the 11 higher order bits of the 32-bit time stamp without any confusion due to wrap-around. Combining these two observations about x, we can throw away the upper 11 bits and the lower 12 bits of the 32-bit time stamps. Therefore, the value of x can now fit in 9 bits taking on an integer value between 0 and 511, as opposed to 0 and $2^{32} - 1$. As a result, σ_x, σ_{x^2}, σ_{xy}, and σ_z are now much smaller.

Using a combination of the above mentioned optimizations, we are able to carry out all the calculation specified in Algorithm.1 using only integers and single-precision floating point numbers.

6.5 McMAC-Light Protocol Description

Having addressed the challenge of synchronization, we now present a simplified version of McMAC that we have implemented on the motes, called McMAC-Light. The goal of the

implementation is to prove that McMAC is a practical protocol even though it requires a certain level of synchronization.

6.5.1 Random Channel Hopping

Time is divided into Small Slots of 1/32768 seconds (i.e. 30.518us) each. The duration of each Small Slot is defined to be the same as the a hardware clock tick for simplicity. Each node keeps a 32-bit counter which increments by 1 every tick. Each Big Slot consists of 128 Small Slots and therefore lasts 3906.3 us. Fig.6.12 shows a Big Slot is divided into 4 periods: channel switching time (12 Small Slots/366μs), guard time(6 Small Slots/183μs), contention window (32 Small Slots/977μs), and data window (78 Small Slots/2380μs).

Channel Switch Time	Guard Time	Contention Window	Data Window (for Data/Probe/Beacon (1440us) + Gap (192us) + Ack (352us) etc.)
12 Small Slots (366us)	6 Small Slots (183us)	32 Small Slots (977us)	78 Small Slots (2380us)

Figure 6.12. Breakdown of a Big Slot.

A node hops at the beginning of every Big Slot in a pseudo-random fashion over all channels. Only the first 128 numbers from the pseudo random hopping sequence are used. The hopping sequence then repeats every 128 Big Slots. The channel switching time of 12 Small Slots allows more than enough time for nodes to hop at the beginning of a Big Slot. A node cannot send or receive during this period. The slot boundaries of different devices need not coincide. A Big Slot begins when the last 7 bits of the local clock become 0. Therefore, by extracting the last 7 bits of the current clock and the seed of the sender node, one can predict its future hopping sequence. Together, we call the 32-bit local clock and the seed the hopping signature of a node. For this implementation, each device has a unique 16-bit MAC address which is used directly as the seed.

A guard time is present right before the contention window to ensure that even though senders not not perfectly synchronized to their receivers, they will not pick a time to start sending before the receiver has switched to the appropriate channel. The contention window allows multiple senders to contend for the medium using CSMA. There is no backoff in McMAC-Light. Data packet transmission starts somewhere within the contention window, and extends into the data window. After a successful unicast packet such as a data packet or a probe packet, the receiver returns an ack packet after a gap of 192μs, as mandated in IEEE 802.15.4 standard[27].

6.5.2 Discovery

Discovery is the process in which devices learn about their current 1-hop neighbors and their corresponding hopping signatures. Every packet sent includes a hopping signature that includes the 16-bit seed of the sender (same as the MAC address) and the current 32-bit local clock. These values are stored in the payload of a TinyOS packet encapsulated inside an 802.15.4 frame as shown in Fig.6.13. A Big Slot begins whenever the last 7 bits of the clock equal 0. The 32-bit time stamp stored in a packet corresponds to the local clock value when the start-frame-delimiter (SFD) of a MAC-level frame is sent. Upon receiving a packet, the receiver can immediately predict the future hopping sequence of the sender. The same 32-bit time stamp is also used for synchronization as described in Sec.6.4.

802.15.4 PHY & MAC Header							TinyOS Header		McMAC Light Header & Payload	802.15.4 PHY & MAC Footer
Preamble	SFD	Frame Len	Frame Control Field	Data Seq No.	Dest. PAN Addr.	Dest. Addr.	Type	Group	McMAC	Frame Check Seq.
4 Bytes	1	1	2	1	2	2	1	1	28 (padded if necessary)	2

Figure 6.13. Frame format of a McMAC beacon or probe packet encapsulated inside an 802.15.4 frame.

802.15.4 PHY & MAC Header / Footer				
Preamble	SFD	Frame Len	Frame Control Field	Frame Check Seq.
4 Bytes	1	1	2	2

Figure 6.14. Frame format of an 802.15.4 ack packet.

When a device first turns on, it announces its presence by broadcasting beacon packets periodically with its hopping signature while hopping on its home sequence. Each node starts broadcasting beacon packets after a random delay between 0 to 1 seconds of power-up. After sending each beacon, it sets another timer to beacon again T_b seconds plus a randomized delay between 0 and 3.9ms (i.e., between 0 and 127 small slots) later.

When a beacon packet is received, the receiver notes the local time and the node ID of the sender. This signature is added to the neighbor table to allow the receiver to predict the sender's hopping sequence and send packets to it. If a neighbor has not been heard from after a timeout period, the neighbor is removed from the neighbor table and the synchronization table.

6.5.3 Synchronization

Synchronization refers to the process in which a node estimates and compensates for the difference in their clocks. Nodes rely on the beacon packets, not only for initial node discovery, but also for synchronization. A node keeps a 32-bit counter ticking at 32768Hz. Each incoming beacon packet contains the time stamp of the sender. When received, the receiver time stamps it using its local clock. Pairs of such corresponding sender-side and receiver-side time stamps are fed into the recursive estimation algorithms for time synchronization presented in Sec. with all the optimizations in Sec.6.4.5.

6.5.4 Rendezvous and Scheduling

Rendezvous is the process in which a sender sends a packet to a specific receiver given that the sender has an accurate estimate of the current home channel of the receiver. Rendezvous can happen on multiple channels simultaneously among several pairs of senders and receivers. A multi-channel MAC protocol optimized for throughput should schedule packets destined for different neighbors carefully to prevent head-of-line blocking and to improve performance. Since we only aim at proving that McMAC is practical, the scheduler uses a simple First-Come-First-Served (FCFS) policy.

At the beginning of a slot, if a data packet arrives at the sender, the sender first checks the neighbor table to find out when the next Big Slot of the receiver starts. Second, it picks a Small Slot within the contention window of this Big Slot. Third, it changes its channel to the home channel of the receiver during this Big Slot. It then waits until the desired Small Slot. Right before sending, the sender senses the medium for a carrier to check that there is not a nearby node sending. If so, it cancels the pending packet and returns to its own home channel. Packets are not retried if the medium is busy in McMAC. If the packet is successfully sent, it waits for an ack from the receiver. The ack packet is generated by the hardware and follows the 802.15.4 standard ack frame format as shown in Fig.6.14. There are no TinyOS or McMAC-Light information in an ack packet.

Each node also sends out beacon packets. Beacon packets are sent periodically on the home channel of the sender. When a beacon packet is due, the sender picks a Small Slot uniformly and randomly within the contention window of its upcoming Big Slot. It waits until the chosen Small Slot, senses the medium, and sends the beacon if the medium is idle.

6.6 McMAC-Light Evaluation

6.6.1 Time Sync Experiments (Exp.T)

In the first set of experiments denoted Exp.T, we want to measure the effectiveness of the time synchronization mechanism in a realistic setting. We setup N nodes in a 1-hop network. They hop over M channels according to their own random hopping sequences. Hopping is suspended when an incoming packet is detected. A node resumes hopping when it has received either (i) a complete packet if that is a broadcast packet or that packet is destined for itself, or (ii) a complete header of a unicast packet if the packet is not destined for itself.

Each node sends out beacons at 1s or 5s intervals. Each beacon packet contains the sender's node ID and the sender's time stamp $txTime$. Beacon packets are not retried if the medium is busy at the time of transmission. In addition, each node sends a probe packet to a randomly chosen neighbor every second. These probe packets are not useful in this set of experiments. They will be used and explained in Sec.6.6.2 to follow. These same traffic pattern is used for both experiments to ensure consistency. In fact, except for the different instrumentation code, the nodes are running the same code in both experiments.

When a node receives a beacon, it uses the receive time stamp ($rxTime$) to estimate the sender's send time ($estiTxTime$). The estimation error is $|txTime - estiTxTime|$. Since we know that clocks do not drift more than 30ppm apart, to combat outliers, a node checks the estimation error and accepts it only if the estimation error is within the maximum drift possible since the previous packet received from this sender. For each accepted beacon, the node sends $\langle rxTime, estiTxTime, txTime, error \rangle$ tuple over USB to a PC for logging.

The accuracy of time synchronization depends on the frequencies of received beacons which also depends on the number of beacons each node is able to send. At the end of the experiment, each node reports the total number of beacons it is able to send.

Because we add 6 ticks of guard time near the beginning of each Big Slot after each hop, a node can tolerate errors within 6 ticks in the clock estimation without any error in the calculation of a neighbor's home channel during the transmission contention window. However, an error of over 6 ticks does not necessarily imply an error in the estimation of the receiver's channel during this window. This is because a node hops only once every 128 ticks.

The following Tab.6.6.1 shows the results for an 1-hour experiment. All numbers except *maxerror* are averages over the number of nodes N.

Experiment ID	T.1	T.2	T.3	T.4	T.5	T.6
Num of Nodes (N)	16	8	4	16	8	4
Num of Channels (M)	8	8	8	8	8	8
Beacon Interval (k)[sec]	1	1	1	5	5	5
Max error [ticks]	94	26	18	46	23	19
Mean Error [ticks]	1.097	1.072	1.020	1.522	1.222	0.990
Std. Dev. of Error [ticks]	1.489	1.418	1.366	2.590	1.905	1.482
% with <= 3 tick Error	98.72	98.82	99.27	91.14	94.27	97.26
% with <= 6 tick Error	99.96	99.95	99.88	96.94	98.98	99.45
# Beacons sent	3429	3457	3453	696	690	698
# Beacons accepted	5153	2451	1067	755	355	182
# Beacons accepted per neighbor	343.5	350.2	355.5	50.3	50.8	60.8

Table 6.2. Results of McMAC-Light time synchronization experiments (Exp.T).

As expected, the estimation errors decrease as the beacon frequency increases. The fact that estimation errors increase as the number of nodes increases is unexpected.

6.6.2 Channel Tracking Experiments (Exp.C)

In the second set of experiments, we want to evaluate the combined effectiveness of the synchronization and channel tracking mechanism to allow senders to find their receivers under light load. A direct measure of this effectiveness is the fraction of the number of unicast packets sent that are received correctly.

This experiment builds on the previous experiment. In addition to the beacon broadcasting in the sync experiment every k seconds, each node sends a probe packet every j seconds to a randomly chosen neighbor. The probe packets are mock data packets. As in the sync experiment, to avoid repeated collisions, each node sends a probe packet every j with a random delay between 0 and 127 small slots (i.e., 0 to 3.9ms).

All nodes enable hardware acknowledgement generation, known as *autoack*. The hardware automatically generates an ack frame after receiving an incoming frame with a correct CRC, a matching destination address, and with the acknowledgement request flag set. After sending a probe packet with autoack enabled, the sender waits a period of 75 ticks (i.e., 2.29ms) for the ack frame. At the end of this timeout or at the reception of the ack frame,

whichever occurs first, the sender logs whether the transmission was successful and whether an ack is received. Neither beacon nor probe packets are retransmitted.

In principle, since each probe packet, like any packet, contains a time stamp of the sender, it can allow any node overhearing it to synchronize with the sender. Therefore, any probe or data packets can act as beacon packets. This would reduce control overhead of our protocol. However, our particular radio platform requires autoack be enabled only if hardware address filtering is also enabled. Hardware address filtering disallows overhearing of unicast packets. To allow fastest and simple ack generation, we decided to use hardware autoack. Thus, only the intended receiver will accept the probe packet; other nodes that overhear the probe packet will not receive it. There are potentially other ways to generate acks quickly while allow overhearing, but we leave it as a future enhancement since the results show that synchronization works well enough with the use of beacon packets alone.

Experiment ID	C.1	C.2	C.3	C.4	C.5	C.6
Num of Nodes (N)	16	8	4	16	8	4
Num of Channels (M)	8	8	8	8	8	8
Beacon Interval (k)[sec]	1	1	1	5	5	5
Probe Interval (j)[sec]	1	1	1	1	1	1
# sent probe packets	27127	13570	6794	18194	9824	4681
# received probe packets	25501	13080	6590	17362	9516	4614
# acked probe packets	25001	12772	6503	16870	9367	4540
% received / sent	94.01	96.39	97.00	95.43	96.87	98.57
% acked / sent	92.16	94.12	95.72	92.72	95.35	96.99

Table 6.3. Results of McMAC-Light channel tracking experiments (Exp.C).

The results of this set of experiments are shown in Tab.6.6.1. The results follow the trends of the time synchronization experiments in 6.6.1 with more frequent beaconing slightly increasing the chance that a node receives a probe from its sender. Senders are able to find their receivers over 94% of the time. There were some ack packet losses on the order of 2-3%, but overall, over 92% of probe packets are acked. There are several reasons why a receiver fails to receive a probe packet, which we can improve on:

1. **Synchronization Error** If the sender is not synchronized with the receiver well enough, it can miscalculate the receiver's current channel and cause a probe packet to be lost. However, this is not the most likely reason when the nodes beacons every second. As we see in the previous section, nodes are synchronized over 99% of the time in such case.

2. **Receiver Busy Sending** The receiver has to send probe packets to others as well, so it might have deviated from its home channel. This will cause the sender to be unable to locate its receiver correctly.

3. **Receiver Busy Receiving** After receiving a packet, the receiver has to move the packet from the buffer in the radio into its main memory of the micro-controller. During this time, another packet cannot be received.

4. **Receiver Deviation from Home Sequence** A node pauses hopping when the beginning of a packet is received. If a node receives a packet near the end of its Big Slot, it will prevent it from hopping to the channel of its current big slot.

There are at least 3 reasons why ack packets are lost:

1. **Noise** The ack packets are corrupted by random noise.

2. **Collision** There is a gap of 12 symbol periods (i.e., $192\mu s$) between the probe frame and the ack. It is possible that another sender starts transmitting during this gap and causes a collision.

3. **Receiver Hops Away** By the time the autoack frame is sent, the receiver has crossed big slot boundary and has hopped away from the channel on which the probe frame was received. However, this would happen only if the receiver started receiving the probe frame near the end of its big slot, which is probably caused by the sender's being poorly synchronized to the receiver.

6.7 Summary

Using off-the-shelf sensor network hardware, we have shown that it is possible to synchronize 1-hop neighbors as the nodes beacon periodically following random hopping sequences. The hardware platform we use is very limited in terms of computational power and memory capacity. The operating system we use does not support real-time operations and the clocks we use has a coarse granularity of only about $30\mu s$. The radio is not optimized for high speed hopping either. Despite all these constraints, we show that 1-hop neighbors can synchronize to within $200\mu s$ 97% of times. This level of pair-wise synchronization is enough for the purpose of parallel rendezvous multi-channel MAC protocol such as McMAC.

As the design of radio improves, we expect the channel switching time to go down to further benefit parallel rendezvous protocols. We also expect in a typical hardware

platform optimized for throughput (rather than power consumption as is the case for sensor networks), one would use a more powerful micro-controller to control the radio with the following effects:

- **Higher Throughput:** our current system is throughput-limited by the speed of the micro-controller, not by the radio. We expect this bottleneck to be removed.
- **Tighter Synchronization:** one major source of time synchronization error is the error in time stamping of incoming and outgoing packets. Using a faster micro-controller will reduce interrupt handling time. Further, it will enable the use of real-time OS to further reduce the jitter in handling time stamping interrupts, resulting in more accurate time stamps.

We therefore expect parallel rendezvous protocols to be a very practical way to increase the network throughput of an ad hoc wireless network in which each node has only 1 radio and where the traffic pattern is more or less uniform among neighbors.

Chapter 7

Summary

7.1 Future Work

As mentioned in Sec.1.3, we focused our study on single hop wireless networks carrying only best-effort traffic. There are also a number of simplifying assumes we made. In this section, we mention a few of such areas where further work is required to bring McMAC into reality.

Since McMAC was designed to use only pair-wise local synchronization, extending the network to multi-hop networks does not affect the ability of neighbors to track the clocks and hopping sequences of their neighbors. However, there are other issues that might come up:

- **Hidden Node Problem** Hidden nodes may exist in a multi-hop network. In such case, immediately after switching to a new channel, a sender waiting to send may listen for a carrier and thinks the medium is idle when, in fact, another node is sending. If the sender starts sending, it may interfere with an existing conversation. The severity of the hidden node problem depends on the carrier sense sensitivity of the sender and the fading/shadowing environment. In general, if the carrier sense range is significantly larger than the communication range of a node, this problem is less severe.

- **Relaying** A multi-hop network must support some form of end-to-end relaying. Relaying can be done either at the network layer using an ad hoc routing protocol or at the link layer using a forwarding scheme. Routing at the network layer is usually more flexible and can use any available wireless links. Forwarding at the link layer usually restricts itself to choosing links on a subset of links such as a spanning tree.

Forwarding using only a spanning tree is much easier to implement because packets travel either to or from the root of the tree. Whether to use network layer or link layer relaying is still an open issue for McMAC.

- **Relay Scheduling** Regardless of the exact forwarding mechanism used, a packet received at an intermediate node must be re-sent to the next hop quickly to reduce the end-to-end delay. While the receiver is forwarding the packet to the next node along the forwarding path, the original sender must not try to send another packet to this receiver. Without any coordination, this process can be inefficient. Since a large fraction of the traffic sent by a node can be due to forwarded packets rather than locally generated ones, optimizing this case is essential to the performance of the network.

There are also a number of other desirable improvements that should be considered:

- **Voice/Video Support** Real-time voice and video are an important class of applications. Voice have a very stringent delay, jitter, and packet loss requirement. Buffered streaming video requires a high throughput guarantee, but delay jitter is unimportant. Real-time video requires both high throughput and low jitter. To effectively support video and voice traffic, McMAC needs to be extended to support constant bit rate (CBR) flows with periodic packet arrivals.

- **Multiple Radios** McMAC currently supports only one radio per node. If a node has more than one radio, there are at least two ways they can be used. One way is to treat the radios as if they were independent. Each radio would have a different seed and hence a different random hopping sequence. When a node beacons, it includes the seeds of all of its radios. To send a packet, the sender uses one of its free radios to send to one of the radio of the receiver. Another way to use multiple radios is to use the same seed for all radios. Therefore, nodes only need to broadcast one seed. When the first radio becomes busy (i.e., either sending or receiving), it deviates from its default hopping sequence. At this time, the second radio turns on to replace the first radio. If a node has multiple radios, it can send and receiver to multiple neighbors at the same time.

- **Broadcast** McMAC does not have any mechanism to support broadcast specifically. One can certainly broadcast the same packet multiple times to try to cover as many neighbors as possible, but this is inefficient if broadcast packets are very frequent.

Another idea is to divide time into two phases: broadcast periods and unicast periods. During broadcast periods, all nodes tune to the same channel for broadcast traffic.

- **Receiver Scheduling** McMAC currently serves receivers in a random order. This is done to avoid deadlocks. However, a random schedule does not have a bound on the maximum service delay which is undesirable. Depending on the objective such as minimizing jitter or maximizing throughput, different scheduling disciplines be used.

7.2 Conclusion

The investigation into designing a multi-channel MAC protocol for wireless networks was based on a simple observation: there are several channels available in an ad hoc 802.11 network, but only 1 is used for the entire network. As the traffic intensity increases, the throughput share of each node decreases very rapidly. Further more, if a flow travels over several hops, the amount of resulting traffic it generates is enormous. The obvious question was, is there a way to use all the available channels? The answer is yes.

On one hand, as shown in Sec.1.2.2, if it is possible to build a radio that makes use of the entire available spectrum, there is no capacity gain by breaking up the spectrum into several channels. On the other hand, regulations and engineering limits often result in the creation of multiple channels. In such case, the use of a multi-channel MAC can potentially increase the total throughput of the network by a factor proportional to the number of available channels, if the traffic is uniform among neighbors. Therefore, multi-channel MAC protocol should be viewed as a complementary software approach to increasing the throughput which rides on top of any advancement in radio hardware design.

Throughout our study, we have focused on the single hop wireless network. Therefore, issues such as routing and relay scheduling were not investigated. However, the McMAC protocol was designed with an eye to extend it to multi-hop networks. Therefore, we specifically avoided requiring global synchronization. Our proposed McMAC should be viewed as the building block for a multi-hop version of the multi-channel MAC protocol.

Before introducing a new protocol, we surveyed the existing literature on multi-channel MAC protocols carefully. The major issue in the design of such protocols is the mechanism nodes use to agree on a channel for data transfer. Since the nodes have only 1 radio, there is a challenge in locating the receiver. We found that many of them share the same principle of operation. We generalized the rendezvous mechanisms into four categories: Dedicated

Control Channel, Split-phase, Common Hopping, and Parallel Rendezvous. Through both analysis and simulation, we studied the pros and cons of each approach and the conditions under which they perform well. Only Dedicated Control Channel uses two radios, the others use one only. It serves as a benchmark for the other schemes.

Apart from Parallel Rendezvous, all three other approaches suffer from congestion in the common control channel. Control channel congestion problem happens when many channels are available or the transfer length after each rendezvous is short. Different approaches suffer to a different extent, but none of them can avoid this problem.

When control channel congestion does not happen, the Dedicated Control Channel usually performs the best among the three. Under this approach, each node tunes one of its two radios to the control channel permanently, and hence nodes have up-to-date information about which channels and nodes are busy. This gives it performance advantage when each node needs to communicate with a large number of neighbors. Moreover, it does not require any synchronization. The downside of this approach is that each node must have 2 radios.

The advantage of Common Hopping is that it requires only one radio per node. However, after each successful rendezvous, the sender reserves the medium for a fixed amount of time even if it does not have enough data to fully utilize the entire duration. This allocated duration grows linearly with the number of available channels. As a result, as the number of channels grows, the wastage also increases. Common Hopping also requires very tight global synchronization.

The Split-phase approach is very simple to implement and requires only very coarse time synchronization. However, its performance is very sensitive to the duration allocated to the control phase versus the data phase. Furthermore, this approach does not work well when each node needs to communicate with a large number of neighbors. Control channel congestion is also the worst using this approach.

Parallel Rendezvous protocols shows good promise in terms of data throughput and the ability to effectively use many channels. The major difficulty with Parallel Rendezvous is the requirement of tight synchronization among neighbor pairs.

To prove that Parallel Rendezvous is a viable approach, we designed McMAC using this approach. The guiding principles of our design are: (i) generality over optimality and (ii) robustness via randomization. We specified the full McMAC protocol state machines, and chose a subset of features for implementation. The implementation serves two purposes: to prove feasibility and to discover any hidden complexity.

Through the implementation exercise, we learnt that the most difficult challenge in implementing McMAC is the time synchronization among neighbor pairs. Consequently, we designed a synchronization method based on recursive estimation. This method can correct both long term clock drift and short term variation by keeping a few numbers per neighbor. Experimental results show that nodes only need to broadcast once a second to keep the synchronization between nodes to within $\pm 200\mu$s over 99% of the time. If it were not not because of limitations of the hardware platform we use, these beacon packets could be piggy-backed on data packets to further reduce overhead and increase accuracy. Overall, the synchronization and rendezvous mechanism work well together to deliver 19 out of every 20 data packets sent.

In conclusion, our implementation shows that McMAC is practical. There are still a number of desirable features which are not included in the current protocol proposal, namely deterministic delay bounds, bandwidth guarantees, and multi-hop network support. However, the major stumbling block in terms of time synchronization and tracking neighbors random hopping sequence have been resolved and verified through experiments. The synchronization mechanism indeed keep nodes well synchronized even though they hop randomly and the beacons are not guaranteed to arrive periodically. Furthermore, our analysis and simulation both show that McMAC can achieve much higher throughput than single rendezvous protocols by avoiding control channel congestion. As a result, we expect McMAC to perform especially well in real applications when channels and nodes are numerous.

Bibliography

[1] Polarizone Technologies Sdn. Bhd. (600204-U). SDR Design Bench `http://www.polarizone.com/`.

[2] Atul Adya, Paramvir Bahl, Jitendra Padhye, Alec Wolman, and Lidong Zhou. A Multi-Radio Unification Protocol for IEEE 802.11 Wireless Networks. Technical Report MSR-TR-2003-44, Microsoft Research, 2003.

[3] Chipcon AS. CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver Data Sheet (rev. 1.3) `http://www.chipcon.com/files/CC2420_Data_Sheet_1_3.pdf`.

[4] Chipcon AS. `http://http://www.chipcon.com/`.

[5] P. Bahl, R. Chandra, and J. Dunagan. Ssch: Slotted seeded channel hopping for capacity improvement in ieee 802.11 ad-hoc wireless networks. In *MobiCom*, 2004.

[6] J. Chen, S. Sheu, and C. Yang. A new multichannel access protocol for ieee 802.11 ad hoc wireless lans. In *PIMRC*, volume 3, pages 2291 – 2296, 2003.

[7] UC Berkeley EECS 150 Components and Design Techniques for Digital Systems. Data sheets for calinx /calinx2 `http://www-inst.eecs.berkeley.edu/~cs150/sp06/Documents.php`, 2006.

[8] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, December 2002.

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

[10] IEEE. IEEE 802.11a-1999 (Supplement to IEEE Std 802.11-1999), High-speed Physical Layer in the 5GHz Band, 1999.

[11] Atheros Communications Inc. Atheros AR5002G 802.11b/g WLAN solution `http://www.atheros.com/`.

[12] BelAir Networks Inc. `http://www.belairnetworks.com/`.

[13] MeshDynamics Inc. `http://www.meshdynamics.com/`.

[14] Tropos Networks Inc. `http://www.tropos.com/`.

[15] Jinyang Li, Charles Blake, Douglas S. J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of ad hoc wireless networks. In *Mobile Computing and Networking*, pages 61–69, 2001.

[16] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi. The Flooding Time Synchronization Protocol. Technical Report TR No.: ISIS-04-501, Institute for Software Integrated Systems, Vanderbilt University, 2004.

[17] Sunnyvale California-USA Maxim Integrated Products, Inc. Max2820, max2820a,max2821, max2821a 2.4ghz 802.11b zero-if transceivers data sheet rev. 04/2004, 2004.

[18] David L. Mills. Internet time synchronization: The network time protocol. In *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems, IEEE Computer Society Press.* 1994.

[19] Jeonghoon Mo, H. Wilson So, and Jean Walrand. Comparison of multi-channel mac protocols. In *the 8-th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, October, 2005.

[20] S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Commun. ACM*, 31(10):1192–1201, 1988.

[21] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *Proc. of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 25-27 2005.

[22] Ashish Raniwala and Tzi cker Chiueh. Architecture and Algorithms for an IEEE 802.11-Based Multi-Channel Wireless Mesh Network. In *Proc. IEEE INFOCOM 2005*, Miami, Florida, USA, March 2005.

[23] H. W. So and J. Walrand. McMAC: A Multi-Channel MAC Proposal for Ad-Hoc Wireless Networks. Technical report, April 2005.

[24] Jungmin So and Nitin Vaidya. Multi-channel mac for ad hoc networks: Handling multi-channel hidden terminals using a single transceiver. In *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, May 2004.

[25] Jungmin So and Nitin H. Vaidya. A multi-channel mac protocol for ad hoc wireless networks. Technical report, UIUC, 2003.

[26] IEEE Computer Society. *ANSI/IEEE Std 802.11 1999 Edition (R2003) Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.*

[27] IEEE Computer Society. IEEE 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), 2003.

[28] Z. Tang and J. Garcia-Luna-Aceves. Hop Reservation Multiple Access (HRMA) for Multichannel Packet Radio Networks. In *Proc. of the IEEE IC3N'98, Seventh International Conference on Computer Communications and Networks*, 1998.

[29] The TinyOS Project. `http://webs.cs.berkeley.edu/tos/`.

[30] A. Tzamaloukas and J. Garcia-Luna-Aceves. Channel-hopping multiple access with packet trains for ad hoc networks. In *In Proc. IEEE Mobile Multimedia Communications (MoMuC '00), Tokyo*, 2000.

[31] Asimakis Tzamaloukas and J. J. Garcia-Luna-Aceves. Channel-hopping multiple access. In *ICC (1)*, pages 415–419, 2000.

[32] Shih-Lin Wu, Chih-Yu Lin, Yu-Chee Tseng, and Jang-Ping Sheu. A Dynamic Multi-Channel MAC for Ad-Hoc LAN. In *Proc. International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '00)*, page 232, Dallas/Richardson, Texas, USA, December 2000.

[33] Shih-Lin Wu, Yu-Chee Tseng, Chih-Yu Lin, and Jang-Ping Sheu. A Multi-channel MAC Protocol with Power Control for Multi-hop Mobile Ad Hoc Networks. *The Computer Journal*, 45:101–110, 2002.

[34] Markus Zannoth, Thomas Rhlicke, and Bernd-Ulrich Klepser. A highly integrated dual-band multimode wireless lan transceiver. *IEEE Journal of Solid-State Circuits*, 39(7):1191–1195, July 2004.