

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Secure and efficient wireless networks

Permalink

<https://escholarship.org/uc/item/2dr7s4cw>

Author

Bellardo, John Michael

Publication Date

2006

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

Secure and Efficient Wireless Networks

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in
Computer Science

by

John Michael Bellardo

Committee in charge:

Professor Stefan Savage, Chair
Professor Rene L. Cruz
Professor Keith Marzullo
Professor Ramesh R. Rao
Professor Geoffrey M. Voelker

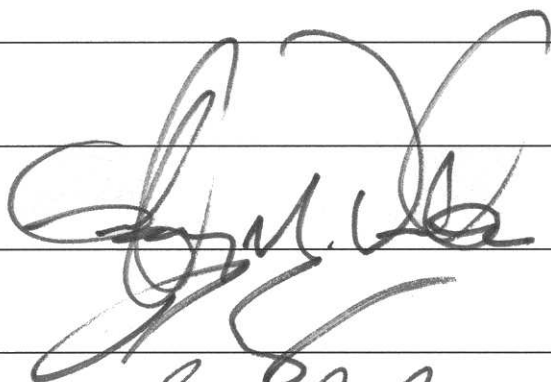
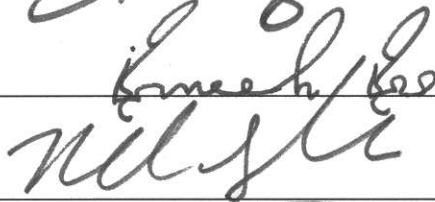
2006

Copyright

John Michael Bellardo, 2006

All rights reserved.

The dissertation of John Michael Bellardo is approved, and
it is acceptable in quality and form for publication on micro-
film:



Stefan Savage Chair

University of California, San Diego

2006

TABLE OF CONTENTS

	Signature Page	iii
	Table of Contents	iv
	List of Figures	vii
	List of Tables	x
	Acknowledgments	xi
	Vita	xii
	Abstract	xiii
Chapter 1	Introduction	1
	1.1. Contributions	2
	1.2. Organization of the dissertation	3
Chapter 2	802.11 Background and Infrastructure	4
	2.1. Introduction	4
	2.2. Network Topology	4
	2.2.1. Identification	5
	2.2.2. Association	6
	2.3. Interference	8
	2.4. Direct Impact of Interference	8
	2.4.1. Direct Interference Terminology	9
	2.5. Indirect Impact of Interference	10
	2.5.1. Retransmission	11
	2.5.2. Deferring	11
	2.5.3. Backoff	12
	2.6. Wireless Testbed	12
	2.7. Installation Characteristics	13
	2.8. Operating System Image	15
	2.9. Image Installation	17
Chapter 3	Automatic Channel Selection in Infrastructure Networks	19
	3.1. Introduction	19
	3.2. Related Work	21
	3.3. Channel Assignment Policies	22
	3.4. Experimental Setup	24
	3.4.1. Testbed Infrastructure	24
	3.4.2. Evaluation Methodology	25

3.4.3.	Impact of uncontrollable load	27
3.5.	Evaluating the Policies	28
3.6.	Race Condition	31
3.7.	Bsync: Ameliorating the Race Condition	33
3.7.1.	Scan Phase	36
3.7.2.	Scheduling Phase	36
3.7.3.	Waiting Phase	37
3.7.4.	Announce Phase	38
3.8.	Evaluating Bsync	38
3.8.1.	Bsync goodput	38
3.8.2.	Convergence Time	39
3.9.	Explaining the Goodput Differences	41
3.10.	Channel Selection Conclusions	44
Chapter 4	802.11 Denial-of-Service Vulnerabilities and Defenses	46
4.1.	Introduction	46
4.2.	Related Work	48
4.3.	Vulnerabilities	50
4.3.1.	Identity Vulnerabilities	50
4.3.2.	Media Access Vulnerabilities	54
4.4.	Practical Analysis of Attacks and Defenses	56
4.4.1.	Deauthentication Attack	57
4.4.2.	NAV attack	62
4.5.	Conclusion	66
4.6.	Special Thanks	66
Chapter 5	Future Directions and Insights	68
Chapter 6	Conclusion	70
Appendix A	Wireless Testbed Manual	71
A.1.	Chapter Outline	71
A.2.	Overview	71
A.3.	Node Specific Information	73
A.3.1.	Hardware Specifications	74
A.3.2.	Power-over-Ethernet customization	74
A.3.3.	Mounting Considerations	76
A.4.	Node Software	78
A.4.1.	Pebble Linux Distribution	79
A.4.2.	Installation Process	80
A.4.3.	Pebble Skeleton Directory	82
A.4.4.	Additional Pre-Built Packages	83
A.4.5.	Additional Custom-Built Packages	84
A.4.6.	Custom-Written Applications	85

A.4.7. BIOS Configuration	85
A.4.8. Custom Software Configurations	86
A.4.9. Custom Status Reporting	88
A.4.10. Customized Atheros Driver Information	89
A.4.11. Capture Portal	89
A.4.12. Initial configuring a new node	91
A.5. Testbed Interconnect	92
A.5.1. Switch Configuration	93
A.6. Wireless Control Machine	95
A.6.1. Connectivity	96
A.6.2. Compilation	96
A.6.3. Services	96
A.6.4. Node Installation Database	97
A.6.5. Other useful management tools	98
Bibliography	99

LIST OF FIGURES

Figure 2.1	General layout of an infrastructure topology.	5
Figure 2.2	A depiction of hidden terminal interference. Nodes C_1 and C_2 can not hear each other and always result in a collision at AP A	9
Figure 2.3	A depiction of exposed terminal interference. Nodes C_1 and C_2 defer to each other's transmissions, however no collision would have occurred because AP's A_1 and A_2 are only within range of their respective clients.	10
Figure 2.4	CSE Building floorplan with wireless node locations depicted. This building comprises roughly 150,000 square feet spread over four floors (and a smaller basement, not shown). Circles indicate testbed nodes, and triangles indicate campus production access points. The basement houses an additional 12 nodes not shown in this figure.	14
Figure 3.1	Goodput as a function of offered load for an average of 10 different random channel assignments. Each value on the x-axis is the number of 1K packets transmitted from a user-level application on the client to its associated AP.	25
Figure 3.2	Overall goodput for two experiments performed 6 days apart using the same channel selection policy.	27
Figure 3.3	Performance achieved by the 802.11a policies when adding one new AP to a preexisting network.	28
Figure 3.4	Performance achieved by the 802.11a policies when used across an entire network.	29
Figure 3.5	Performance for a subset of 802.11b policies while the building was closed over the holidays.	30
Figure 3.6	Performance of the standard 802.11a policies.	30
Figure 3.7	Simultaneous start of the channel selection algorithm as compared to a completely staggered start using the RSSI Max policy in 802.11a. This graph reveals some of the lost performance potential due to the race condition.	31
Figure 3.8	Performance with 5 seconds of random jitter added to the start time in 802.11a. This jitter does not increase overall network performance over the synchronized case.	33
Figure 3.9	The 4 phases of bsync, phase transitions, and time spent in each phase. Note the only phase whose length varies is the waiting phase.	35
Figure 3.10	Diagram showing multiple APs running bsync simultaneously. Circles represent preexisting APs, while stars are new APs running bsync.	36

Figure 3.11	Performance of Bsync with the RSSI Max policy compared with simultaneous and staggered startups for 802.11a.	39
Figure 3.12	Performance comparison for Bsync with the RSSI Sum policy and a subset of the other 802.11b policies taken while the building was closed over the holidays. The Least Occupied policy employed a fully staggered startup.	40
Figure 3.13	Performance lost potential due to underlying 802.11 properties as seen in one 802.11b experiment. The loss is measured in bits wasted per kilobyte successfully transmitted.	43
Figure 4.1	Graphical depiction of the deauthentication attack. It is noteworthy that the attacker needs only generate one packet for every six exchanged between the client and access point (AP).	51
Figure 4.2	Diagram depicting the NAV Attack in action. The gradient portion of the attacker's frame indicates bandwidth reserved by the NAV although no data is actually sent. Continually sending the attack frames back to back prevents other nodes from sending legitimate frames.	53
Figure 4.3	Packets sent by each of the 4 client nodes during the deauthentication attack. The first attack, against the MacOS client, started at second 15 and lasted 8 seconds. The second attack against all the clients started at 101 and lasted for 26 seconds. The attacking node consumes a negligible amount of bandwidth due to the rate limiting.	56
Figure 4.4	Packets sent by each of the 4 client nodes during the deauthentication attack with an access point modified to defend against this attack. The first attack, against the MacOS client, started at second 10 and lasted 12 seconds. The second attack against all the clients started at 30 and lasted through the end of the trace. The attacking node consumes a negligible amount of bandwidth due to the rate limiting.	57
Figure 4.5	Results from the ACK based NAV Attack simulation with 18 client nodes. The attack begins at time 40 and ends at time 60. The dark region at the bottom of the graph during the attack is the attacker.	61
Figure 4.6	Results from the ACK based NAV Attack simulation with 18 client nodes modified to implement defense. The attack begins at time 40 and ends at time 60. The dark region at the bottom of the graph during the attack is the attacker.	63

Figure A.1	CSE Building floorplan with wireless node locations depicted. This building comprises roughly 150,000 square feet spread over four floors (and a smaller basement, not shown). Circles indicate testbed nodes, and triangles indicate campus production access points. The basement houses an additional 12 nodes not shown in this figure.	72
Figure A.2	The location of the ethernet isolation transformer on the Soekris net4826 board. The image on the left shows the whole board, while the image on the right is a close up shot of the transformer. In both images the transformer is circled.	75
Figure A.3	The location of the center tap pins for the ethernet isolation transformer on the Soekris net4826 board. Each of the two pins is circled.	76
Figure A.4	A wire soldered to each of the center tap pins.	77
Figure A.5	The location of the ethernet jack pins on the bottom of the Soekris net4826 board. The image on the top left shows the whole board, the image on the top right is a close up shot of the ethernet jack pins, and the image on the bottom identifies the particular pins in question. The interesting parts of all images are circled.	78
Figure A.6	The wires soldered to the ethernet jack pins on the bottom of the Soekris net4826 board.	79

LIST OF TABLES

Table 3.1	Channel assignment policies considered in this chapter.	23
Table 3.2	Convergence times for the Least Occupied policy across all startup models. Results are from 802.11a with a dwell time of 1 second. Each point is an average of 10 runs.	40

ACKNOWLEDGMENTS

Chapter 4, in full, is a reprint or has been submitted for publication of the material as it appears in the Proceedings of USENIX Security Symposium 2003, Bellardo, John; Savage, Stefan. The dissertation author was the primary investigator and single author of this paper.

VITA

2006	Doctor of Philosophy in Computer Science University of California, San Diego San Diego, CA
2001	Master of Science in Computer Science University of California, San Diego San Diego, CA
2000-2006	Graduate Student Researcher Department of Computer Science and Engineering University of California, San Diego
1999	Bachelor of Science in Computer Science California Polytechnic State University San Luis Obispo, CA
1998-2006	Systems and Network Administrator California Integration Coordinators Placerville, CA
1996-1999	Summer Intern SRI Consulting Menlo Park, CA

PUBLICATIONS

John M. Bellardo and Stefan Savage, “802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions”, *Proceedings of the USENIX Security Symposium*, Washington D.C., August 2003.

John M. Bellardo and Stefan Savage, “Measuring Packet Reordering”, *Proceedings of the 2002 ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

FIELDS OF STUDY

Systems and Networking

Wireless Networks

ABSTRACT OF THE DISSERTATION

Secure and Efficient Wireless Networks

by

John Michael Bellardo

Doctor of Philosophy in Computer Science

University of California, San Diego, 2006

Professor Stefan Savage, Chair

In 1985, the Federal Communications Commission "opened" a range of radio frequencies called the Industrial Scientific and Medical band. For the first time anyone could transmit high-speed data wirelessly without a license and with minimal regulatory restrictions. Building on this opportunity, the IEEE's 802.11 protocol enabled cheap untethered access to nearby wired networks and ultimately transformed how millions of businesses, governments, and casual computer-users access the Internet.

However, the designers of 802.11 did not anticipate the magnitude of their success, and thus the protocol is hard-pressed to meet many of the new demands placed on it. In particular, 802.11's decentralized management structure poses unique challenges for both security and performance. This dissertation focuses on two key instances of these problems: *denial-of-service* and *channel efficiency*.

In 802.11, each wireless client implements the media access control protocol (MAC) to share the common radio medium more effectively while maintaining the decentralized design of the protocol. Additionally, each client implicitly *trusts* that the other clients are faithful in their MAC implementation. Under these best-case circumstances the protocol operates as designed, however, nothing prevents a client from abusing this trust. In this work I evaluate two of the more critical abuses of trust, both of which result in a denial-of-service attack where a well-behaved client is denied access to the wireless medium. In addition, I propose defense mechanisms that successfully

protect the networks while preserving the decentralized nature of 802.11.

The desire to efficiently use all available radio channels arises because typical deployments have many more access points and clients than they do channels. This creates a high degree of contention for a limited resource. The decentralized nature of 802.11 exacerbates this issue because each access point makes a locally greedy channel decision even though the aggregate of the local decisions does not yield a global optimum. To that end I propose and evaluate a number of different automated channel selection policies and synchronization techniques using a large, real world testbed, identifying the current best practice and quantifying its performance improvement.

Chapter 1

Introduction

Wireless data networks, and 802.11-based products in particular, have been one of the fastest growing network technologies in the last decade. In fact, a survey of market analysts performed by the Dell'Oro Group [Gro] places the worldwide market for 802.11 equipment at \$3.4 billion for the year 2006. This popularity is driven by two main factors. Wireless networks provide convenience that has become the de facto standard in developed countries. Society has evolved to the point where continuous connectivity is the norm, not the exception. 802.11 technology has been used, in a large part, to meet the connectivity expectation. To a lesser extent there are a number of circumstances where wireless networks are a more suitable solution than wired networks, while providing sufficiently high bandwidth to satisfactorily support the primary network application.

While there are a number of benefits inherent to this popularity (*e.g.* the low hardware prices this commodity status has created) it is also accompanied by the specter of increased scrutiny. Network operators are discovering the design limitations and complications inherent to the 802.11 specification. For example, several cities, such as Philadelphia [Com] and Mountain View [Blo], are now in the process of providing blanket 802.11 coverage over their entire jurisdiction. Such a large deployment exposes efficiency problems at the networks scale to sizes not anticipated by the 802.11 designers.

However, even more concerning is the prevalence of 802.11 networks used to carry mission and life critical data. For example the health care industry is eagerly adopting 802.11 networks for use with their patient interaction devices, such as heart monitors and intravenous fluid dispensers, to achieve better quality and monitoring capabilities. A recent press release [Net] from Sharp HealthCare [Web] includes the following paragraph on their usage of WiFi:

“We have wireless-enabled intravenous (IV) pumps that automatically gather essential medication information from a central database to ensure the safety of our patients. This allows us to focus more on patient care, rather than administration,” said Wiesenberg. Each smart IV medication system is equipped with an 802.11b PCMCIA card that communicates with a central server that stores medication profiles. These profiles are continuously downloaded to each IV pump to ensure no errors in the distribution of medication to patients. Sharp is also using the wireless environment for bedside charting, electronic medical records, custom pharmacy applications, emergency room admission and ordering, and traditional data center applications.

These mission-critical uses of 802.11 underscore the importance of understanding and improving security vulnerabilities in 802.11.

My thesis addresses key security and efficiency shortcomings in the 802.11 specification. It achieves this through a mix of careful analysis of the specification, instrumentation and measurement of deployed wireless networks, and careful design of solutions to the identified problems. In particular all the resolutions to the problems that I propose have been tested in real networks, are shown to work, and are easily deployed on existing 802.11 hardware today.

1.1 Contributions

In this thesis I address two particular areas of concern in 802.11 wireless networks: availability and efficiency. To improve availability I performed a security analysis of the specification itself, identifying several potential vulnerabilities. Afterwards I ranked the vulnerabilities in order of decreasing severity, based on both the expected

impact and the ease with which the attack can be mounted. I implemented the first two, most potent, attacks and tested them using a controlled, but real, 802.11 network. Based on my experiences mounting these attacks I proposed, implemented, and evaluated defenses for these two attacks. The methodology used in developing these defenses allows them to be easily retrofitted into existing hardware, as well as introducing a general framework for further devising defenses.

Addressing efficiency problems in wireless networks is more involved. To accomplish this I designed and deployed a 200 node wireless testbed inside a building at UCSD. Using this testbed I evaluated initial channel selection algorithms. These algorithms run once when the AP starts up, and then isn't run again unless the access point is restarted. I looked at the achieved "goodput" for a number of common algorithms, and identify the best performer of the group. In addition, I identify a fundamental flaw in those algorithms that negates their added complexity as compared to random channel selection. I propose a new coordination algorithm and protocol to address this problem, and demonstrate that using the algorithm avoids the inherent limitation and restores the performance benefits of the more complex algorithms.

1.2 Organization of the dissertation

This thesis is organized in three main chapters. Chapter two provides the background information on the 802.11 protocol that is necessary to better understand the rest of the work and discusses the more important challenges with the large 802.11 testbed I created at UCSD. Chapter four uses the testbed to examine the impact of channel selection in infrastructure wireless networks, and proposes a new protocol that ameliorates the the biggest problem discovered. Chapter four evaluates availability problems in the 802.11 specification, and proposes tractable solutions to them. Chapter five and six discuss future directions and conclude, respectively. In addition, the appendix provides detailed information on the operation of the testbed.

Chapter 2

802.11 Background and Infrastructure

2.1 Introduction

The research and ideas presented in this thesis require intimate knowledge of the inner-workings of IEEE 802.11 wireless networks. This chapter provides an in-depth review of the 802.11 media access control (MAC) subset that is relevant to understand the research presented. In addition it reviews some of the fundamental aspects of 802.11 networks. The complete IEEE 802.11 specification [Soc99] is available online.

The second part of this chapter is devoted to introducing and discussing the wireless testbed infrastructure that I deployed at UCSD. Detailed information about the hardware and capabilities of the platform are discussed. I also discuss the major obstacles encountered while deploying such a large testbed, and how I solved them.

2.2 Network Topology

Wireless networks were designed to emulate a direct connection to a wired network in order to provide a better experience to the user. However, creating this appearance utilizing a radio interface required a number of abstractions, protocols, and standards. One of the fundamental standards was selecting a network topology. Since this work deals exclusively with the *infrastructure* topology, it is discussed in this section.

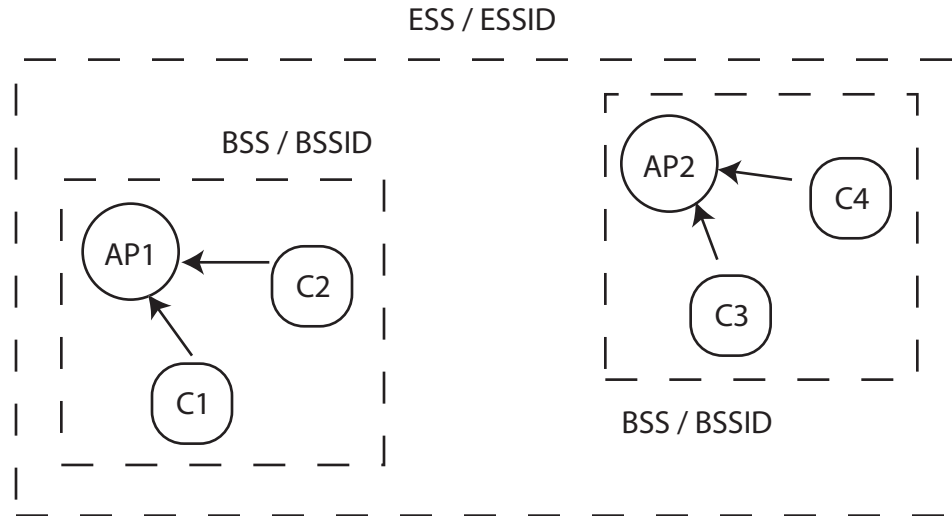


Figure 2.1: General layout of an infrastructure topology.

The infrastructure topology abstracts the broadcast radio medium to create the illusion of a dedicated connection to a wired network. This abstraction is created by installing a bridge node, called an access point (AP), on the wired network. The AP accepts wireless clients and forwards traffic destined for the clients to and from the wired network. Due to the relatively limited range of the radios employed by 802.11, a number of APs must be deployed to provide contiguous coverage over larger physical areas. This density, combined with the unpredictable nature of mobile wireless clients, results in two complementary mechanisms for managing these infrastructure networks: identification and association.

2.2.1 Identification

One important aspect of location is identifying otherwise autonomous APs that belong to the same network. 802.11 achieves this by creating a logical abstraction surrounding each AP that encompasses all the clients it serves. This abstraction is called the *Basic Service Set*, or BSS. Each BSS is assigned an lexicographical identifier by the network operator termed the *Basic Service Set Identifier*, or BSSID. The BSSID assigned by the network operator presumably allows the user identify the correct BSS

(*e.g.* the BSSID used at UCSD is “UCSD”).

When multiple APs are deployed the union of their BSSes is called the *extended service set*, or ESS. The ESS also has an identifier, called the ESSID. Pragmatically the BSSID and the ESSID are the same. In order for a client to identify an ESSID, which is desirable for mobile clients that might associate with multiple APs, all APs in the ESS use the same ESSID. For example, the ESSID at UCSD is “UCSD”. Therefore the specification defines a conceptual difference between the BSS/BSSID and ESS/ESSID even though the only important, user visible attribute, is the ESS/ESSID. Figure 2.1 graphically depicts an infrastructure network with the various service sets and identifiers labeled.

2.2.2 Association

The illusion of a point-to-point link requires the client to be communicating exclusively with an AP, however a client that just entered the coverage area of a networks does not know what APs exist or which one the client wishes to utilize, nor does the AP know which client is using it. To address these issues 802.11 includes a series of management functions that effect an “association” between a client and an AP. This functionality is unique to 802.11 wireless networks, and is necessary to understanding part of the security work presented in chapter 4.

Initially a client has no idea about which APs are within communication range, nor does it know any of the properties that make a particular AP more desirable (*e.g.* the ESSID of the AP). To learn this information it performs a scan of the radio spectrum. This is a procedural scan, where the client first switches to one of 802.11’s channels (*e.g.* 802.11b utilizes 11 channels) and probes for APs. 802.11 defines a special *probe request* frame format for this purpose. All APs that receive the probe request respond with a *probe response* frame. This response frame includes important properties of the AP, such as the actual channel of operation ¹, the BSSID, and the transmission rates

¹Because the channels in 802.11 overlap it is commonplace for a probe request sent on channel 1 to be heard by an AP on an adjacent channel (channel 2 or 3), and the subsequent response to also be heard on channel 1. By including the channel information in the response a client is able to filter out this cross-channel chatter.

supported by the AP.

In addition to responding to probe requests, AP periodically (by default, once every 100 milliseconds) transmit a *beacon frame* that contains the same information present in the probe response frame. This beaconing gives the client an option to perform *passive scans*, where they do not generate a probe request, but still receive all the necessary information. Once a client has completely scanned the spectrum and compiled a list of APs within range it selects one of them. The actual selection criteria varies depending on the purpose of the network deployment, however one rather common criteria is ensuring the ESSID matches a list of preferred ESSIDs (*e.g.*, only associate with “UCSD” and “home”).

Once a client selects a particular AP it is necessary to inform the AP of the selection so it can begin forwarding traffic destined to the client. 802.11 accomplishes this through a two step process. The first step is *authentication* and the second is *association*. Separating these two steps allows a client to be authenticated at multiple APs simultaneously, which reduces the amount of time required for a mobile client to move an association to a different AP.

Authentication allows a client to prove its legitimacy to the AP. 802.11 provides a generic framework for this, ostensibly to allow any number of different authentication algorithms. In addition to the framework the standard defines two particular instantiations, which have become the only two used in practice. The first is open authentication, where an AP simply accepts all clients that attempt to authenticate². This is the most common form used in networks. The second form uses a key, distributed out of band, known by the client and the AP, to establish that the client is authorized to use the AP’s services.

After authenticating a client still must inform the AP it is going to use the AP to bridge to the wired network. It does this through a process known as association. Authentication is a prerequisite for association. After a client has scanned the network, authenticated, and associated with an AP, it gains access to the wired network connected

²In practice some APs use an Access Control List (either black or white) to deny authentication attempts even though they employ “open” authentication

to the AP.

In addition to the association and authentication mechanisms there is also a deauthentication and disassociation mechanism. These are not requests that can be granted or denied, and therefore serve an informational purpose. A client can inform the AP it is no longer authenticated (or associated), which permits the AP to clear any internal state used by the client. Likewise an AP can inform the client it is no longer authenticated and inform it of such so the client knows it needs to reestablish an association.

2.3 Interference

The work in chapter 3 goes to great lengths to minimize the impact of interference on the performance of 802.11 networks. As such it is necessary to have a good understanding of the sources and problems created by interference in 802.11 networks. The impact from interference falls into two broad categories, direct impact (*e.g.*, the transmission was not received) and indirect impact (*e.g.*, the transmitter is required to wait a certain amount of time before attempting retransmission). This section details the impact of both these sources.

2.4 Direct Impact of Interference

The cost of direct interference on the performance of the network is clear. The frame is lost, and another mechanism must be used to recover it. However, direct interference also has the same impact if the explicit acknowledgment for a frame is lost, and there are two terms related to direct interference that are important to understand.

First, 802.11 employs a positive acknowledgment scheme whereby each *unicast* frame (that is, a frame with only one intended wireless recipient) is acknowledged by the recipient upon successful reception. Direct interference can effect either the transmission of the data frame or the transmission of the acknowledgment frame. In either case the result is the same – the transmission is considered a failure.

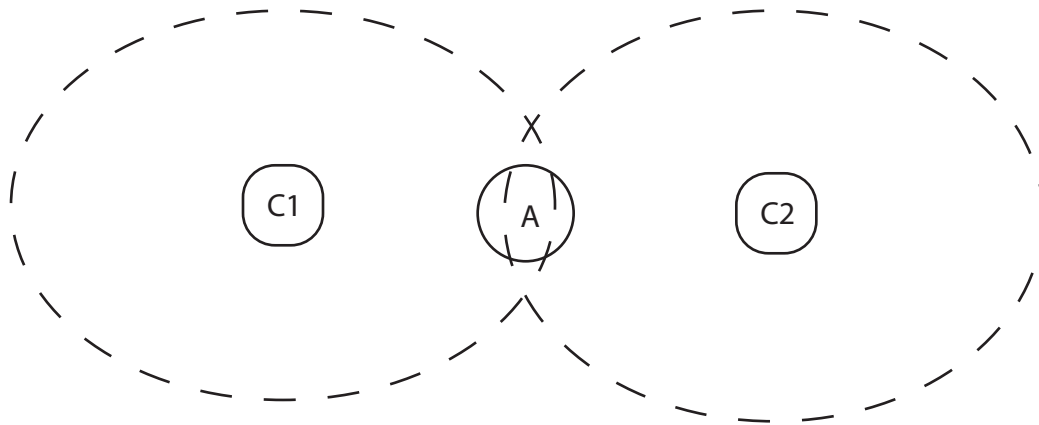


Figure 2.2: A depiction of hidden terminal interference. Nodes C_1 and C_2 can not hear each other and always result in a collision at AP A .

2.4.1 Direct Interference Terminology

There are three common terms that describe the source of direct interference, hidden terminal, exposed terminal, and broadband. Hidden terminal interference occurs when two nodes that can not detect each others energy interfere. Consider the situation where there is a client C_1 , an AP A , and a client C_2 physically located in a line in that order. Furthermore C_1 and C_2 mutually can not detect each other. If C_1 and C_2 both attempt to transmit to A at the same time the transmissions will collide and neither may get through. This is the hidden terminal problem and is depicted in figure 2.2

The exposed terminal problem occurs when a node, again C_1 in our example, defers to another node, C_2 , even though the two transmission could have occurred simultaneously without causing a collision. This is typically the case when C_1 and C_2 are communicating with nodes that are not within range of each other, as depicted in figure 2.3.

It is important to note that channels in 802.11 overlap. The 802.11b signal occupies approximately 22 Mhz of spectrum centered around the channel's middle frequency. Since channels are spaced 5 Mhz apart there is significant overlap. For example, channels 1 and 2 share approximately 17 Mhz of the same spectrum. It is also impor-

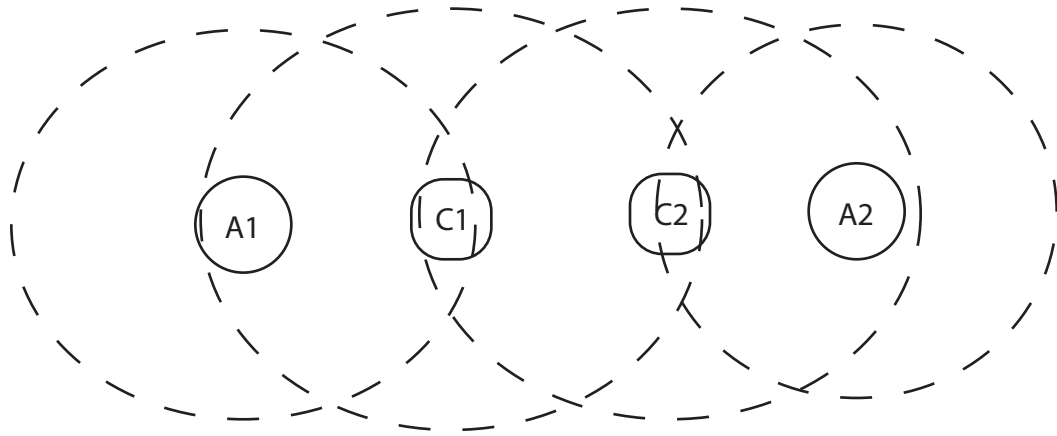


Figure 2.3: A depiction of exposed terminal interference. Nodes C_1 and C_2 defer to each other's transmissions, however no collision would have occurred because AP's A_1 and A_2 are only within range of their respective clients.

tant to realize that the radio signal does not cease to exist after the aforementioned 22 Mhz spacing. It is only attenuated to the point where the probability of causing interference is minimal. However it is still possible for frames transmitted in channel 1 to be successfully decoded in channel 11. Some work refers to interference from nodes on the same channel as *co-channel* interference and different channels *adjacent-channel* interference.

Broadband interference is a generic term that captures all interference from sources that don't understand the 802.11 protocol. Common man-made sources of broadband interference are microwaves, radars, and baby monitors. Additionally nature provides sun spots and other phenomena that have the potential to interfere with radio transmissions.

2.5 Indirect Impact of Interference

The indirect performance impact of interference can be much more costly and prolonged than the direct impact. For example, as a result of detected interference on two consecutive transmissions, a wireless node may decrease its transmission

rate, which drastically reduces the performance of the network. The rate may not be increased for many seconds. This section explains the important 802.11 *media access control* (MAC) rules, and how they contribute to the indirect impact of interference.

2.5.1 Retransmission

As mentioned previously, 802.11 uses positive acknowledgments (ACKs) of all unicast frames. In addition to these ACKs, the 802.11 MAC can retransmit a failed frame a small number of times (the actual number is vendor specific, but 7 is frequently used). While detection (*e.g.*, the ACK isn't received) and retry typically happen faster than if left to higher layers, much of the added performance impact is only present when frames are retransmitted.

2.5.2 Deferring

802.11 employs carrier sense in an attempt to avoid interfering with other nodes. Before attempting transmission a node must observe silence on the radio for a period of time. The exact length is one of three values, depending on the collision status and type of the frame being transmitted. These values are collectively called *interframe spacing*.

When transmitting an ACK or *clear-to-send* frame, the node must wait for the *short interframe spacing* (SIFS) period. When attempting the first transmission of any other frame it must wait a *DCF interframe spacing* (DIFS) period. When retransmitting a frame following a previous unsuccessful transmission of the same frame it must wait an *extended interframe spacing* period (EIFS). The exact values for these periods vary between 802.11a and 802.11b, however the relationship $SIFS < DIFS < EIFS$ always holds. Additionally the EIFS is significantly greater than the DIFS, making retransmissions more costly.

If, during the waiting period, another transmission is detected the countdown timer for that period is suspended. This state is known as *deferred* in the specification. Deferring means the period counter is not reset when another transmission is detected.

2.5.3 Backoff

A node performs backoff under two circumstances. First, when another transmission is detected during the DIFS period preceding the original transmission of a frame, and secondly for a MAC retransmission of a failed frame. Backoff is performed by creating a number of fixed length time slots after the DIFS (or EIFS) period. A node initially picks a random slot and must wait until that slot is reached before transmitting. This waiting period is in addition to the DIFS period and obeys the same deferral rules discussed above.

There is one exception to backoff. If no other transmission is heard during the DIFS period immediately preceding the first transmission of a frame, a node is allowed to transmit that frame in the first time slot without selecting a random slot and waiting.

When a failed transmission attempt is detected, in addition to waiting an EIFS period as discussed above, the number of time slots (not the length of a slot) is doubled. The exponentially growing number of time slots creates an exponentially growing waiting period before the next retransmission can occur. This quickly dominates the cost of multiple retransmissions.

The indirect cost of interference, through the combination of retransmissions, deferring, and backoff as found in 802.11, can be quite high. In some cases it can waste tens of milliseconds for a single transmission, time which could have been used transmitting more data instead of continually making no progress on the same frame. This, coupled with the change in 802.11 transmission rate explained at the beginning of this section, can substantially worsen the impact of direct interference that otherwise would have been considered a minor nuisance.

2.6 Wireless Testbed

The existing installed base of 802.11 equipment is valued in the billions of dollars, and by some estimates, is still experiencing phenomenal sales growth. Therefore any work whose goal is to improve the inherent deficiencies in the protocol needs to be

acutely aware of how the proposed solutions will actually operate in real deployments.

With this constraint in mind I performed a series of exploratory experiments aimed at increasing the relevance of my results. The experiments consisted of running a series of small simulations using topologies I could easily recreate in the lab. The same data was collected from the test topology and compared with the simulated results. Unfortunately, due to the richness of a real wireless environment and relative lack of expressiveness in the simulators, I was not able to achieve reasonable results from the simulator, even after multiple refinement iterations.

This experience is not unique to my work. In [KNG⁺04] Kotz *et.al.* debunk the six most common assumptions found in the most common simulators. Some of the identified assumptions include circular transmission ranges of radios, link symmetry, and perfect delivery probabilities for links that do exist. As a result of this work, and my experiences with wireless simulators, I decided to use large scale wireless experimentation as the foundation of my thesis work. In order to run these experiments I had to build and deploy a large wireless testbed.

I initially approached building an experimental as a straightforward engineering problem, however, after two years of operational experience, my opinion has changed. There are good reasons beyond simple funding constraints³, for example the operational complexity necessary to maintain the entire system, that our deployment is the single largest academic 802.11 testbed that I am aware of.

2.7 Installation Characteristics

Size is one of the key attributes of our wireless testbed, in two senses. Our testbed contains a large number of radios, 180, spread over a large indoor space, 150,000 square feet. Figure 2.4 shows the deployment locations for four of the five floors covered in the Computer Science and Engineering building at the University of California, San Diego.

³In fact, the entire testbed cost somewhere on the order of \$60,000, which is not an inordinate amount of money given current grant funding levels.

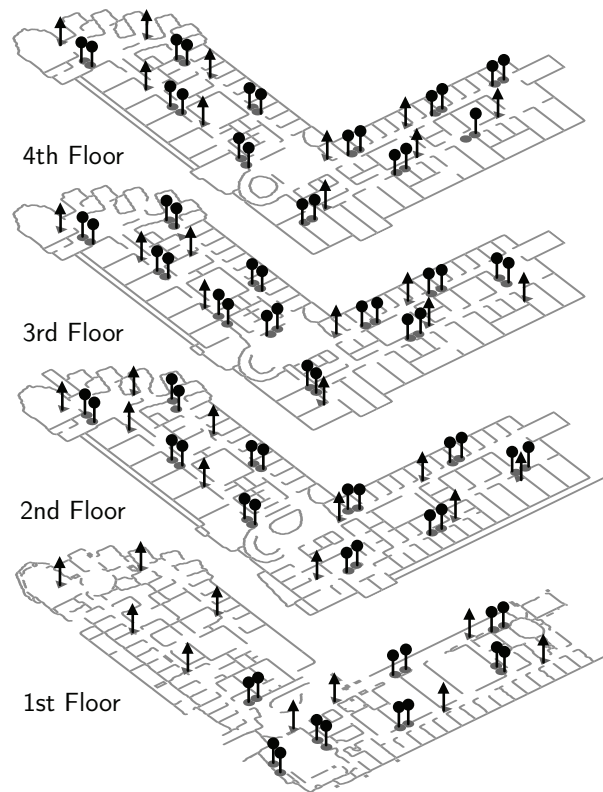


Figure 2.4: CSE Building floorplan with wireless node locations depicted. This building comprises roughly 150,000 square feet spread over four floors (and a smaller basement, not shown). Circles indicate testbed nodes, and triangles indicate campus production access points. The basement houses an additional 12 nodes not shown in this figure.

Each node in our testbed contains two independent radios. This has a number of benefits. It enables research into multi-hop topologies, it enables measurement-based research projects that monitor two channels at the same point in space simultaneously, and it reduces the overall hardware costs of our infrastructure. It does, however, also have a drawback. Simultaneous transmissions from both radios are more likely to interfere with each other due to the close proximity, and the average distance between nodes is greater.

In addition to placing two radios in every node, I installed nodes in pairs (referred to as pods). Before installation I performed a series of bandwidth measurements to determine the minimum spacing necessary between two nodes in a pod without undue intra-pod interference. The nodes are installed as close to this minimum as conditions permit. This pairing provides passive measurement research projects the ability to monitor up to four different channels from what is effectively a single point in space. This is enough coverage to monitor the traditionally non-overlapping channels in 802.11b plus one channel in between.

Each node is directly connected to a dedicated wired network, which in turn connects to the building's network. The nodes are powered using Power over Ethernet. This reduces the installation complexity, expense, and improves the visual aesthetics.

In order to support the widest array of research, each node is an embedded computer that uses a Pentium class CPU and has 64 megabytes of non-volatile FLASH memory and 128 megabytes of RAM soldered on the logic board. The limited amount of unremovable non-volatile FLASH memory, coupled with the size of the testbed, led to the biggest problems in deployment.

2.8 Operating System Image

Even though the specifications for the testbed nodes appear adequate the amount of FLASH memory is a big constraint. It is difficult to fit a modern operating system (OS) and all the necessary supporting applications into 64 megabytes of storage.

The standard operating system sources for desktop machines (*e.g.*, Redhat, FreeBSD, etc.) result in OS installations in the Gigabytes, while embedded distributions are small but lack many of the necessary features. This situation creates a tradeoff between fitting everything within the 64 Megabyte limit and what functionality the resulting image possesses. Through my experience addressing various aspects of this tradeoff I developed a few key criteria for the OS image.

Most importantly the image needs to be easily customizable, because there are a number of features that are necessary to conduct the underlying research. Building all the required software packages by hand is one extreme that represents the ultimate in flexibility, however compilation for this embedded environment is tricky as discussed below. A better solution is being able to install prebuilt packages. Unfortunately most prebuilt packages contain files that are unnecessary for the operation of the program (*e.g.*, documentation and header files). The extra files can constitute a large fraction of the disk space required to install the package.

My solution to this problem was a combination of using preexisting packages and building programs by hand. In order to use the package I had to manually identify all the unneeded files and dependencies, and then write a cleanup program that removed them after installation. Due to the difficulty in compiling programs by hand, I only took that approach when no acceptable package was available. Even when compiled myself it was still necessary to identify and remove unneeded files.

The ability to update an OS image as newer software becomes available is also important. The updates typically involve security fixes and new features, both of which are important. The update process for prebuilt packages is generally streamlined by the package provider, however once the unneeded files have been removed most of the update scripts fail. To solve this problem updates are achieved through rebuilding and reinstalling the entire OS image. This process is entirely automated and discussed in the next section. Updates to software built by hand are more involved, because they require downloading a newer version of the source code and replicating the build. As a result those programs don't get updated as frequently.

Another important feature of the OS image is stability. Once an image has been constructed and tested, it is important for the features and interfaces present in the image to remain the same. This minimizes the amount of work required to maintain all the custom software for each wireless experiment. This stability is achieved through stringently evaluating software changes proposed for the image.

The ability to develop new software and compile existing packages for the image is important, however is it more difficult to achieve. Standard procedure for compilation has the compiler running within the exact environment it is compiling for, however with only 64 Megabytes of memory available for the OS image, it is not possible to even install a compiler. This forces the use of an external environment for compilation.

For our testbed this external environment is a desktop computer with the same OS image installed on it. Unfortunately once the compiler, related files, and other tools are installed the image begins to drift apart from the image on the testbed nodes. Over time this version mismatch between the shared libraries on the compilation machine and the testbed nodes increases. This mismatch can be so severe that it precludes compiling some software packages. This is the primary reason compilation can be difficult, and prebuilt software packages are the preferred method of installing functionality.

The image selected for the testbed was a small Linux distribution called Pebble. Pebble employs the Debian package management system. I added all of the aforementioned infrastructure to pebble. Unfortunately the resulting structure is brittle, and OS updates are still rather difficult to apply.

2.9 Image Installation

After a suitable image is created, getting it installed on the machines is a challenge. The FLASH memory, where the OS image needs to be installed, is soldered on to the logic board. This eliminates the possibility for a normal installation, where the flash is removed, placed in a desktop computer equipped with a flash writer, and the OS imaged installed. Instead a network installation procedure is necessary.

The predictable and reasonably well documented steps of the process start by configuring DHCP to provide the appropriate auxiliary information fields that allow PXE, the network boot loader, to locate a valid configuration. Other parts of the process include configuring a TFTP server, which PXE uses to load the kernel and configuration files, writing the configuration files themselves, and compiling the kernel so that it supports using NFS for the root file system. This whole process, when coupled with a bootable image on a network file server, boots the nodes into a full-feature unix environment.

The installation script uses this environment to install the actual OS image onto the flash resident in each node. Unfortunately this is another area where good, standardized support is lacking. As a result I had to create my own custom and complex installation scripts. The end result of this whole process is what amounts to a custom linux distribution.

Despite the well known shortcomings in existing wireless simulators and their related models, simulation results continue to constitute the bulk of wireless research. In contrast, the work presented in this thesis is grounded in experimental design, implementation, and measurement. In order to achieve those goals I needed to install a large scale testbed at UCSD.

During the course of the installation I discovered why these networks and resulting research are not more prevalent. I've presented the high level challenges and my solutions to those problems in this chapter. Appendix 1 provides more in-depth detail about the operation of the testbed.

Chapter 3

Automatic Channel Selection in Infrastructure Networks

3.1 Introduction

The success of 802.11 technology has greatly increased the density of network deployments – either by design, in the case of enterprise networks designed to maximize coverage, or serendipitously in high-density urban housing. Moreover, overlaid on these deployments are independent commercial “hotspots” and new municipal area networks all sharing the same unlicensed frequency bands.

With this increase in density comes an increased need to carefully manage shared spectrum resources. In particular, each 802.11 access point (AP) can independently select among a set of partially overlapping “channels” within a particular frequency range (2.4Ghz for 802.11b and 802.11g and 5.2Ghz for 802.11a). Thus, each AP’s selection of channel defines a coverage *cell* in which its clients may communicate. Dense network deployments have spatial overlap between these cells (sometimes intentionally to eliminate coverage dead zones) that also creates the opportunity for interference. Since performance is inversely related to interference and contention, differences in channel assignments can ultimately create significant differences in user experience. This is particularly apparent in urban deployments where a single apartment building

may have 10's of APs all using the same factory-default channel.

The problem of channel assignment is not addressed in the 802.11 specification, and as a result a number of best-practice techniques have evolved. For example, in engineered networks, enterprise administrators try to manually minimize the possibility of interference by carefully selecting channels from the so-called “orthogonal” sets (e.g., 1, 6 and 11 for 802.11b).¹ However, such manual assignments are ad hoc, time consuming and error prone and can only reflect a particular static view of the network (sometimes based on a set of initial measurements termed a “site survey”). Thus, there has been considerable interest in automating this process in such a way that operational overhead is minimized and performance is maximized.

Enterprise equipment vendors have attempted to address this problem through various proprietary techniques. Unfortunately, since they are proprietary, the only publicly available information about them is in the form of marketing literature. Given the goals and general lack of accountability of this information, it tends to grossly overstate performance and doesn't provide the intimate details necessary for a good comparison.

This chapter address this problem in the context of two key components. First, using large-scale experimental measurements, I consider a range of automatic channel selection policies that are common in either modern APs or contemporary literature, and compare the quality of the assignments they generate. Second, I identify and explore the impact of unintended synchronization that occurs in when multiple APs employ these policies independently. I define a distributed synchronization approach that provides significant improvements in overall performance in a real network setting. Finally, I analyze the underlying reasons for these improvements and their implications for future work in this area.

¹In fact such channels are not truly orthogonal in any formal sense, but their separation is sufficient to provide a minimum attenuation for a given physical layer implementation (e.g., for 802.11b the spectral mask guarantees that a signal will attenuate by at least -50dB over 22Mhz). This is generally sufficient to minimize the probability that such a signal will overwhelm a competitor, although it may still “interfere” in the sense that other transmitters may defer to its transmissions.

3.2 Related Work

The channel assignment problem has long been studied in the wireless context, most commonly as an application of graph coloring which has a large theoretical literature including proofs of hardness and practical approximations [HS65, KMS94, ALM⁺92, SLM92]. However, a pure graph coloring representation lacks the expressiveness to capture the richness of the wireless environment (particularly interference and attenuation) which has led to a variety of more specialized models typically including weighted graph algorithms or their integer programming equivalents. These algorithmic approaches are discussed in relation to 802.11 in [LKC02, Moh], while [KWBF03] and [MMS03] explore the same problems in the context of cellular networks.

More concretely, several papers describe *centralized* schemes for making channel assignments based on measures of interference. Brik et al. describe a hypothetical system in which a central server tracks all aspects of available spectrum based on reports from clients [BRBB05]. Servers then provide frequency “leases” to clients (using a dedicated control channel) to minimize overall interference. A less radical proposal is provided by Mishra *et al.* who propose a system that learns *range* sets and *interference* sets based on proposed 802.11k client measurements [MBB⁺06]. Using this information a centralized server then assigns channels to APs. Using simulations driven by empirical measurements they show that such assignments are likely to improve overall throughput. Finally, Raniwala *et al.* explore similar algorithms in the context of multi-hop multi-radio mesh networks [RGcC04, RcC05].

Recently, Mishra, *et al.* [MBA05] also explores distributed channel assignment in infrastructure networks. They propose an abstract metric, the *I-value*, which attempts to better model interference characteristics as a graph coloring parameter. They provide approximation algorithms that minimize the I-value and show that these produce assignments with improvements in abstract quality metrics (e.g. number of conflicting edges in a graph). Finally, Raju *et al.* describe small-scale measurements of two channel selection policies (similar to the “least occupied” and “RSSI Max” policies described

later) and show performance improvements in an ad-hoc networking context.

My work is distinguished from previous efforts largely by its empirical nature. Rather than rely on simulation, I perform large-scale measurements of a 160 node network and explore the real performance impact of different channel selection policies in this environment. Moreover, I build a distributed channel selection algorithm and demonstrate that it is practical and effective in this context. Finally, I use analysis of empirical measurements to explain *why* our policy leads to better performance.

3.3 Channel Assignment Policies

In this section I present the channel assignment algorithms I study in this chapter. I categorize channel assignment algorithms according to the mechanisms and policies they employ. The mechanism determines how an AP acquires information about its radio environment (*i.e.*, does it scan or randomly sample), and the policy dictates how this information is used to select a channel. Table 3.1 summarizes these policies.

The **Fixed** policy is the most basic. In this policy all APs select the same channel. It corresponds to the deployment of APs using a default configuration. The majority of consumer APs ship from the factory with a pre-programmed channel, and individual users typically neglect to change this setting. This interaction makes the Fixed policy the de-facto standard for consumer equipment, even though it is intuitively the worst.

In the **Random** policy, each AP selects a channel uniformly at random on which to operate. This policy is easy to implement and avoids the pervasive channel conflicts of a Fixed policy. However, due to the limited number of non-overlapping channels in environments like 802.11b, this policy will not scale well to dense deployments.

Scaling to denser deployments presumably requires policies that take into account their radio surroundings, such as the operating channels of neighboring APs. In the subsequent policies, APs actively scan all channels for fixed intervals and record

Table 3.1: Channel assignment policies considered in this chapter.

802.11b Policies	802.11a Policies
Fixed	Fixed
Random	Random
Unoccupied	Unoccupied
Least Occupied	Least Occupied
RSSI Sum	RSSI Sum
RSSI Max	RSSI Max

information about wireless activity on the channels. Policies then use the information obtained by scanning channels to make channel assignment decisions.

The **Unoccupied** policy selects an unoccupied channel. An unoccupied channel is one in which no beacons from other APs were observed during the scanning interval.

The **Least Occupied** policy also selects among unoccupied channels. If no such unoccupied channels exist, however, Least Occupied chooses the channel with the fewest other APs observed.

When choosing among occupied channels, the Unoccupied and Least Occupied policies consider each AP equivalent in terms of the impact of sharing a channel. In practice, however, the degree of interference is influenced by the signal strength of the interfering AP at the node being interfered with. As a result, I consider two additional policies based on the signal strength of other APs measured while scanning.

The **RSSI Sum** policy treats interference effects as additive, and therefore adds together the signal strengths (RSSI) for each unique AP operating on a channel to create a synthetic channel “score”. If more than one signal strength measurement is available for a given AP, which occurs when multiple beacons are overhead in the measurement period, all the measurements for that AP are averaged before summing into the score. The AP selects the channel with the lowest score.

Similarly, the **RSSI Max** policy assigns a score to each channel and selects the channel with the lowest score. This policy, however, uses the maximum RSSI value measured on a channel as the score. The intuition being that the AP with the largest

signal strength will dictate the channels performance characteristics.

Note that, to avoid clearly poor selections, all policies except Fixed and Random by default avoid overlapping channels when possible. In the event of a tie for the “best” channel, a policy removes all the standard overlapping channels from consideration. For example, in 802.11b, the channels 2,3,4,5,7,8,9, and 10 are removed. If, after removal, there are no channels left, all overlapping channels are added back into the set for consideration.

Even after filtering out the overlapping channels it is still possible that multiple channels tie for “best”. A particular policy implementation is free to break this tie in any technique. All of the implementations studied in this work break the tie by selecting the first available channel. The impact of this choice is evaluated later in this chapter.

3.4 Experimental Setup

In this section I describe the testbed infrastructure used for performing realistic experiments, and the methodology for evaluating channel selection policies.

3.4.1 Testbed Infrastructure

All the experiments for this work were performed using the wireless testbed described in chapter 2. In order to measure network performance it is desirable to place a known and repeatable load on the network. This requires some of the radios in the testbed to act as clients. However, due to the physical proximity of the two radios inside a single node, it is not possible to have one act as a client and the other an AP without noticeable interference. This occurs because of cross talk between the boards, regardless of the channel separation between the radios. A straightforward solution would be to use one radio in a pod ² as an AP and another radio, located in the other node in the pod, as a client. Unfortunately that reduces the total AP count in the experiment from 80 down to 40, too small to model a large infrastructure networks.

²Recall there are two nodes per pod, each of which has two radios

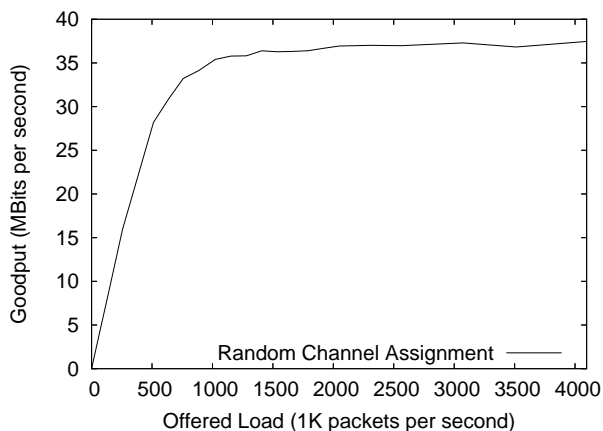


Figure 3.1: Goodput as a function of offered load for an average of 10 different random channel assignments. Each value on the x-axis is the number of 1K packets transmitted from a user-level application on the client to its associated AP.

Therefore I settled on a third, compromise solution. Each node in the testbed operates as an AP and participates in the channel selection process, which provides the scale necessary for this work. When measuring the performance of a channel assignment only a subsection of the network is checked. This is akin to sampling from a larger population. This sampling does not bias the results because none of the channel selection algorithms are aware of which subset is going to be measured.

To sample the network performance I deployed nine additional nodes in research labs and faculty offices in one wing of the building on the third floor. These nodes emulate client devices that connect to the testbed APs, and enable the use of controlled workloads between clients and APs as the basis for evaluating channel selection. The hardware and software configuration of these nodes closely matches that of the testbed nodes, which facilitates collecting the same set of statistics at both ends of the network connection.

3.4.2 Evaluation Methodology

I performed all experiments using both the 802.11a and 802.11b frequency ranges. When using 802.11a, I used the eight 802.11a channels dedicated for indoor

use. Our testbed nodes are the only devices operating in the 802.11a frequencies in the building, therefore I have sole use of these channels when performing experiments.

Unfortunately for these experiments the building where the testbed is located also has a deployed 802.11b access network. This “production” network increases the uncertainty of the 802.11b experimental result. Section 3.4.3 discusses how the production access network interacts with our experiments in more detail.

To exercise the network for a particular channel assignment, I used two user-level applications running on the client nodes and the APs. A source application on the client transmits packets to its associated AP and tracks the number of bytes it sends. A sink application on the AP then tracks the number of bytes actually received. The number of bytes received for the duration of an experiment measures the “goodput” between the client (source) and AP (sink). At both the client and AP, I recorded various statistics, such as the number of collisions, that provide insight into the underlying variations in performance observed during an experiment.

The traffic pattern for these experiments was a constant offered load from each client. In the experiments, each client sends 1K UDP packets at a constant rate to the APs. Figure 3.1 shows the relationship between network goodput and offered load for a random set of channel assignments. For this particular channel assignment policy, the goodput peaks around a rate of 1500 packets per second. Unless otherwise noted all experiments used an injection rate of 4096 packets per second. This value, chosen to be substantially higher than the 1500 mark found in the previous experiment, ensured that the network is saturated.

A number of the policies I evaluate rely on randomness. To reduce the effects of variability from random choices, I average the measured goodputs across 5–10 different channel assignments generated by the same policy. For a given set of policies, I generate all of the assignments in close time proximity to each other. Doing so exposes every algorithm to roughly the same variations in the wireless environment. For example, when comparing the Fixed, Random, and Unoccupied policies, I would generate one assignment using each policy in quick succession, and then repeat the process to

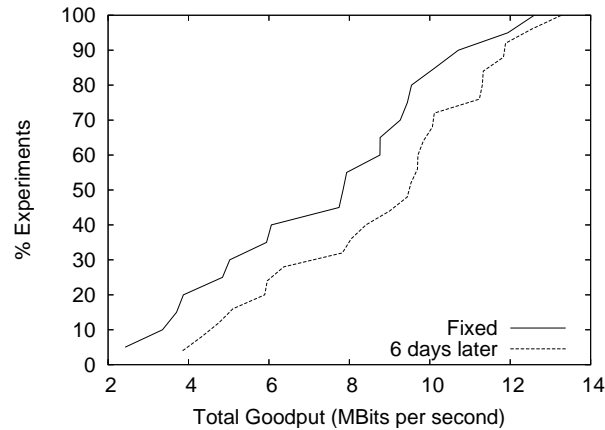


Figure 3.2: Overall goodput for two experiments performed 6 days apart using the same channel selection policy.

create the second, third, *etc.* assignments.

The associations between clients and APs are predetermined and fixed in order to remove potential variability introduced as channel assignments change. Each measurement period lasts for 60 seconds, and, again, were performed cyclicly to minimize the impact of changes in the wireless environment. I measure the goodput for an individual assignment at least 5 times to account for minor variations in measurement. The net result of this process is a set of between 25 and 50 goodput measurements for each policy.

3.4.3 Impact of uncontrollable load

802.11a and b networks operate in two different parts of the spectrum, each of which has unique attenuation and propagation characteristics. One of the goals of this work is to examine the behavior of the policies in these different environments to determine whether a single policy works best in both spectrums, or if different policies are required.

As mentioned previously, the wireless testbed is deployed in a building that already has a production 802.11b access network. I ran all 802.11b experiments during the evening and night hours to avoid the heaviest usage periods. However, there is still

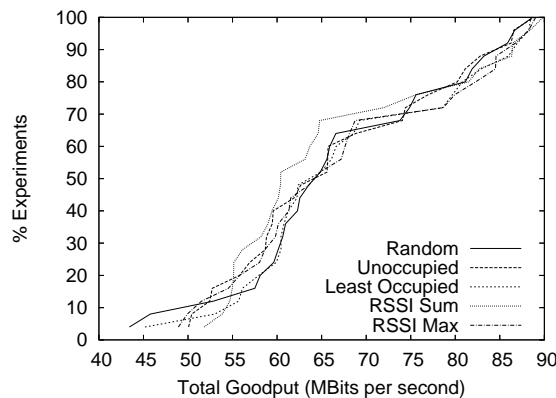


Figure 3.3: Performance achieved by the 802.11a policies when adding one new AP to a preexisting network.

a large variation in these results, sometimes greater than 25%. Figure 3.2 shows the observed performance difference for the same channel selection policy taken approximately a week apart. Unfortunately, the variation of 25% as seen in the graph introduces too much noise into the 802.11b results for them to be conclusive. However, the bulk of the results reported in this chapter are based on measurements taken using 802.11a, where no other devices are operating and the environment is pristine. In addition, there are some 802.11b results from the 2005 holiday break when the building was officially closed and ostensibly vacant. While these 802.11b results do not paint a complete picture of all the policies, they do provide insight and are discussed as appropriate.

3.5 Evaluating the Policies

To begin, I consider the scenario of a single AP being added to an existing 802.11 environment. This corresponds to an individual user installing a new AP in an apartment building where neighbors already have APs of their own, or a reset of a single AP in an enterprise infrastructure network (e.g., due to configuration change or firmware upgrade).

To eliminate the probability of a “straw man” experiment (e.g., all other APs are operating with a poor channel selection) I first create a high-quality assignment for

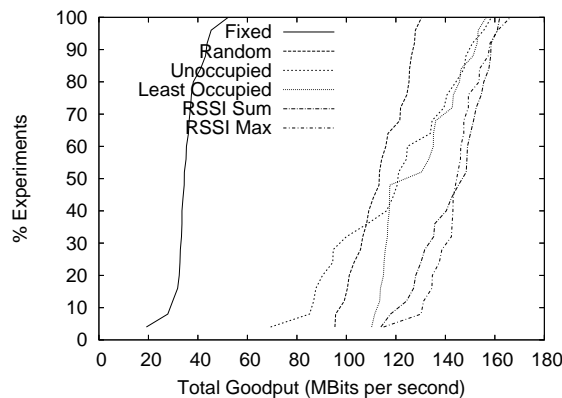


Figure 3.4: Performance achieved by the 802.11a policies when used across an entire network.

our testbed using the best performing algorithm identified in this chapter (RSSI Sum with Bsync synchronization as described later). Given this “baseline” assignment, one of the APs was then instructed to select a new channel, using the policy being evaluated, and a series of goodput measurements are taken between clients and their associated APs as described in the methodology section 3.4.2. Figure 3.3 plots the CDF of these data points on a per-policy basis, referred to as the goodput CDF, for all the policies tested. From the graph, it is clear the selection policy makes relatively little difference when a single AP is added to an already existing network.

The various policies better differentiate themselves when all APs in the network employ the same policy. Figure 3.4 shows the resulting goodput CDF under this scenario. For this experiment all APs were started at the same time. Surprisingly all the policies besides random perform similar to the fixed policy. This indicates the added complexity of the other policies is both unnecessary and detrimental.

802.11b networks also exhibit a similar pattern of behavior. Figure 3.5 shows similar data for 802.11b, however due to the time constraints when the building was closed, only a subset of the policies were evaluated. It is clear that the counter intuitive result for the random algorithm holds in the 802.11b frequency range.

The explanation for this behavior is that the tie breaking rule is used more than the normal policy, and therefore dominates the performance of the channel selec-

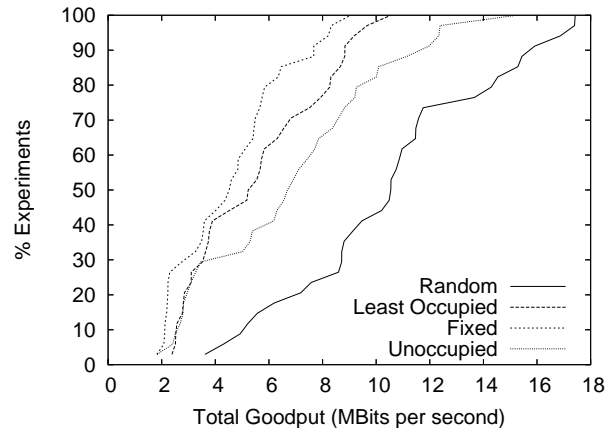


Figure 3.5: Performance for a subset of 802.11b policies while the building was closed over the holidays.

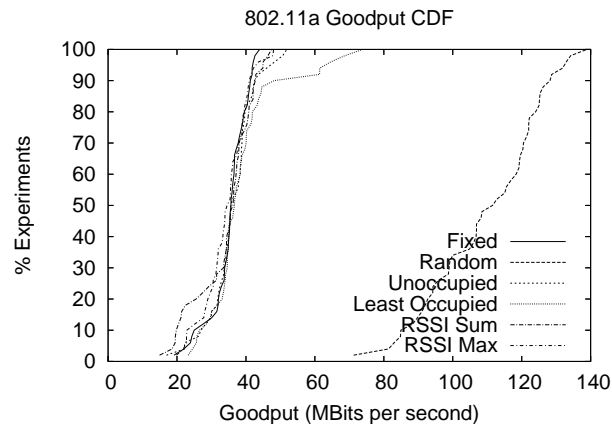


Figure 3.6: Performance of the standard 802.11a policies.

tion algorithm. Recall that the exact tie breaking rule was left to the discretion of the implementation, and that all the implementations in this work pick the lowest channel when there are ties. This tie breaker is very similar in function to what is commonly found in equipment: pick the first channel in the list of best channels.

The following example better illustrates the problem. Two APs passively scan the spectrum at roughly the same time. Since both are passively scanning neither one hears the other. Also assume there are no other APs nearby. After both APs complete their respective scans all channels have the same score, hence the tie breaking rule se-

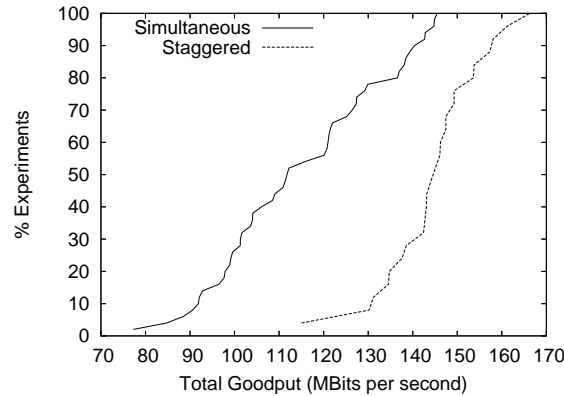


Figure 3.7: Simultaneous start of the channel selection algorithm as compared to a completely staggered start using the RSSI Max policy in 802.11a. This graph reveals some of the lost performance potential due to the race condition.

lects a channel. If the rule deterministically picks a channel, either by selecting the lowest channel or the first in the list, both APs end up operating on the same channel and interfering with each other.

To verify this explanation I changed the policies to use a random tie breaker and re-ran the same set of experiments. Figure 3.4 shows the goodput CDF with a random tie breaker. Random is no longer the clear winner, however it still performs respectably. Clearly reverting to a random tie breaker is not an ideal solution because some performance benefit is lost. However, before the situation can be more adequately addressed, it requires a more detailed investigation.

3.6 Race Condition

When multiple APs are turned on simultaneously, two subtle but related problem surface. First, during the scanning phase, two nearby APs will never detect each other because they are both passively scanning. Since both APs are scanning the same number of channels at the same rate they completely miss one another. Second, it is extremely likely that the AP that finishes its scan first will select a channel that the other

AP has already scanned. Since the channel was already scanned the information that the first AP is operating in that channel will be excluded from the information the second AP uses to make its channel selection. Since both APs were scanning at the same time, it is also extremely likely that the radio conditions observed during the scan are similar. Therefore, when the second AP makes its selection it is likely to pick the same channel the first AP did. For example, the least occupied channel policy is very similar to “fixed” in such situations.

To evaluate the potential impact this synchronization has on performance I ran another set of experiments controlling the startup time explicitly. Channel assignment on the APs were centrally controlled such that only one AP was selecting a channel at a time. For example, AP_1 was allowed to completely finish its channel selection algorithm before AP_2 started. I refer to this as *staggered* startup. The other scenario, where all APs begin their channel selection at the same time, is referred to as *simultaneous* startup. Figure 3.7 compares performance of the RSSI Max policy under both the staggered and simultaneous startup models. The graph shows a performance difference of over 30 MBps, on average, between the two models. In essence, there is significant performance potential that cannot be realized when the startup times are simultaneous.

The race condition is predicated on the assumption that APs get started at the same time. However, in our experience this is a entirely realistic assumption due to synchronized reboots. The three main reasons for an operational AP rebooting after deployment all lead to this problem: bulk firmware upgrade, bulk programming changes, and building power failure. The first two are artifacts of centrally managed networks, where updates are usually processed with automated tools in bulk. The third reason, power outages, can occur in apartment buildings as well as centrally managed networks.

A common approach to remove the impact of synchronization in distributed protocols is to simply add random jitter to the process. Thus, each AP could select a random delay before initiating their own channel selection process, thereby avoiding a race with other APs. However, this “solution” is deceptively simple and is highly sensitive to the choice of *how much* delay to use. For example, Figure 3.8 illustrates

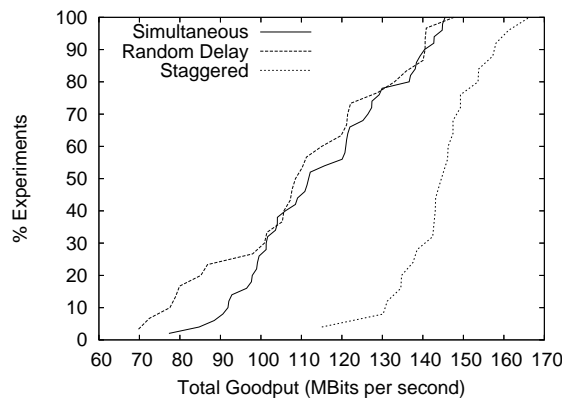


Figure 3.8: Performance with 5 seconds of random jitter added to the start time in 802.11a. This jitter does not increase overall network performance over the synchronized case.

that adding a random delay of 0-5 seconds to each AP makes no perceivable impact on goodput. This is due to the transitive nature of the channel selection process – one AP’s choice of channel may impact all other AP’s decisions. Thus, to be effective the range of jitter values must be large enough that no two APs should expect to overlap in their scanning phase. Unfortunately, the exact size of this value is a function of the size and deployment of the network and is difficult to calculate in advance. A value large enough to be sufficient for dense deployments would be overbearing on small and medium networks, and a happy medium value wouldn’t work well for large networks. Finally, for many networks this approach can introduce large delays (in excess of 10 minutes) before the overall network is ready for use.

To address this problem I developed an explicit synchronization protocol, called *Bsync*, that schedules channel selection among APs.

3.7 Bsync: Ameliorating the Race Condition

Section 3.6 described the race condition present when APs are simultaneously powered up, and showed how this can be mitigated by staggering AP startup time. The staggered startup model, however, is impractical for two reasons. First, it requires a

centralized scheduler to order each AP's channel selection times – particularly difficult in deployments with multiple administrators such as office or apartment buildings. The second problem is the amount of time required to converge on a complete channel assignment. For the 802.11a policies tested in this chapter the staggered start model took approximately 980 seconds to converge in our testbed. To address these two concerns I propose a distributed synchronization algorithm, called *bsync*.

In addition to eliminating the channel selection race condition, the design is motivated by two other goals. First, *bsync* must operate completely over the wireless network interface. This allows the algorithm to work in dense networks that cross administrative boundaries as are commonly found in office and apartment buildings. The second goal is that *bsync* delivers an acceptable convergence time. Unfortunately, as explained before, this second goal is at odds with perfectly solving the race condition.

Optimally addressing the race condition in this scenario, with no constraints on when an AP can join or leave the channel selection process, is provably impossible via a reduction to asynchronous consensus [Lam96]. An optimal, perfect solution notwithstanding, most nodes in a dense infrastructure network will not be able to directly communicate with each other over the radio, and the time complexity inherent in multihop synchronization would be excessive. Therefore, my approach assumes each node only makes local decisions based on overheard broadcast traffic and short timeout values. This allows forward progress at a reasonable rate, and as this chapter will show later, reasonably solves the race condition problem.

The *bsync* algorithm is divided into four distinct phases as follows: Scan, Scheduling, Waiting, and Announce. Figure 3.9 illustrates these phases, related phase transitions, and how much time is spent in each phase from the perspective of a single AP participating in *bsync*. Figure 3.10 depicts the interaction between multiple APs running *bsync* simultaneously. These two figures, in combination with the detailed protocol description that follows, completely explain *bsync*'s operation.

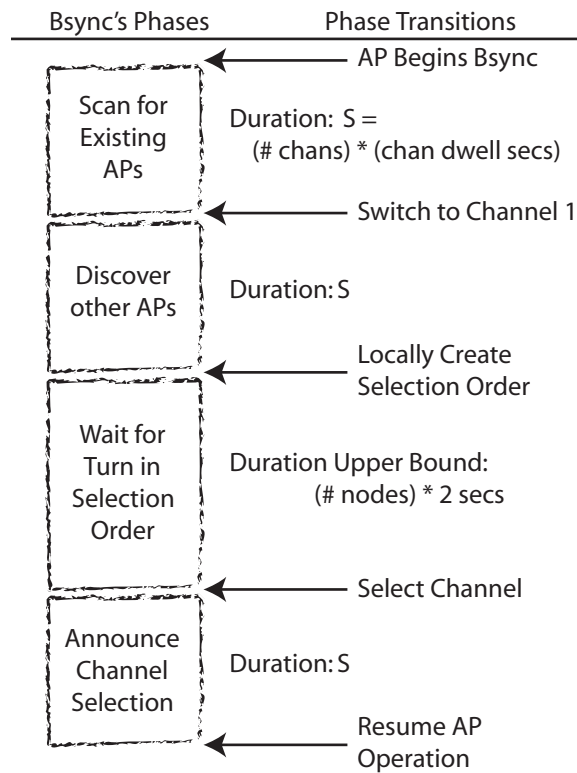


Figure 3.9: The 4 phases of bsync, phase transitions, and time spent in each phase. Note the only phase whose length varies is the waiting phase.

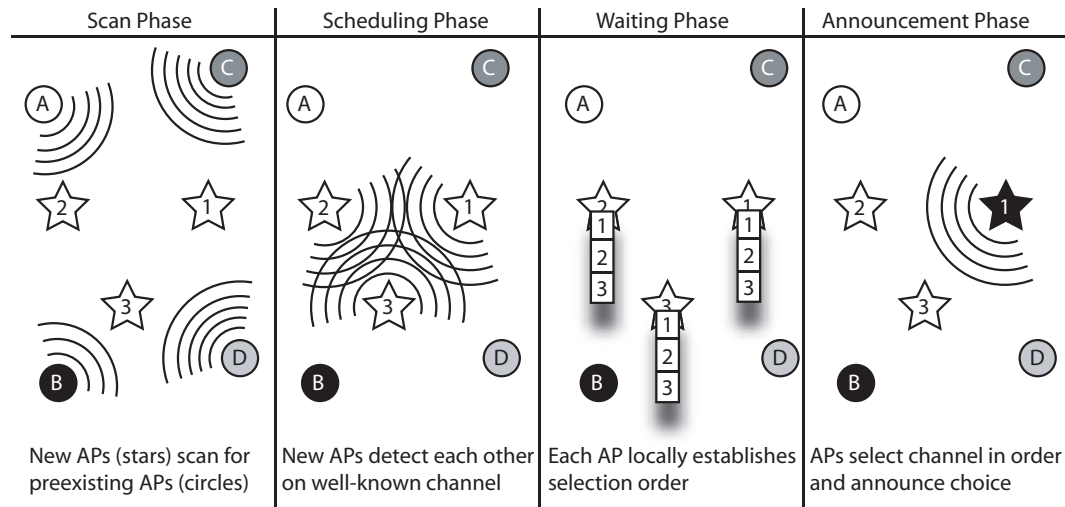


Figure 3.10: Diagram showing multiple APs running bsync simultaneously. Circles represent preexisting APs, while stars are new APs running bsync.

3.7.1 Scan Phase

The goal of the scan phase is to learn about other APs that are already operating within range of the current AP. This is accomplished using the same strategy employed by other scanning algorithms. Each candidate channel is visited in turn, listening for beacons from other APs. Basic statistics are tracked for each AP another AP hears, including the number of beacons it sent, the average signal strength, and the channel of operation. At the end of the scan phase, an AP has discovered other nearby APs, but has not yet learned about nearby APs also scanning the spectrum.

The amount of time spent in this phase is a function of the number of channels scanned and the channel dwell time (how long one waits to hear beacons on a particular channel). I refer to this scan duration as S , defined by $S = dwell * channels$. Once the scan is complete the AP transitions to the scheduling phase.

3.7.2 Scheduling Phase

The scheduling phase permits APs participating in bsync to discover each another. At the beginning of this phase an AP switches to a well-known channel. In

the implementation this was the first channel in the spectrum range. Once operating in the channel the AP beacons as if it were an operational AP. These beacons provide both the mechanism for transmitting the bsync synchronization data and the algorithm's mnemonic lineage.

Each beacon frame is augmented to indicate that the AP supports bsync and to provide additional information key to the protocol – the ultimate choice of channel and a timeout value (described below). Each AP receiving one of these beacons notes this information for later use.

An AP participates in the scheduling phase for S seconds, the same amount of time spent in scanning during the first phase. This duration was chosen to compensate for small amounts of jitter in the actual start times of the bsync algorithm, as expected when startup is nearly simultaneous. Once this time has elapsed an AP moves to the waiting phase.

3.7.3 Waiting Phase

Before starting bsync's third phase, the waiting phase, an AP creates a local scheduling order. It does so by sorting all the "friendly" APs in increasing numeric order by MAC address. After sorting it assigns a timeout value of 2 seconds to each other AP. An AP leaves this phase when its MAC address is at the top of the list. However, while it is waiting, it continues beaconing. The data field in these beacons contain the amount of time remaining in the waiting phase, assuming the AP must wait for all timeouts.

During this waiting period an AP is also listening to the other beacons. If another AP reports a channel selection it is duly noted and that AP is removed from the schedule. Additionally, if it overhears another decision timeout value, the local AP updates its timeout value to reflect the other AP's countdown. In this way, APs typically wait long enough to accommodate the selections of other APs they can't directly communicate with. Clearly this does not result in perfect synchronization, however the next section demonstrates the synchronization is very good. Once all the previous APs have either timed out or selected a channel, an AP moves to the announce phase.

3.7.4 Announce Phase

This is perhaps the simplest phase. Before entering this phase an AP uses all the statistics gathered in the first three phases and selects a channel based on its policy. Then, for S seconds, the AP announces its channel selection by indicating the frequency (in Mhz) of its selected channel in the beacon frames it sends on the well known coordination channel. This announcement allows other APs after it in the selection order to update their internal accounting to reflect the channel selection, and then proceed to select their channel as appropriate. Once this phase is completed the AP switches to the selected channel and begins normal AP operation.

3.8 Evaluating Bsync

In this section I evaluate Bsync according to its goodput performance for clients, and its convergence time to produce a channel assignment.

3.8.1 Bsync goodput

I used a similar experimental setup to evaluate Bsync as with the previous experiments. I implemented the Bsync mechanism and measured its performance for the complete set of policies. Figure 3.11 compares a single 802.11a policy, RSSI Max, to both the synchronized and staggered startup models; Bsync was only tested using the synchronized model. Bsync performs substantially better than the simultaneous model, but not quite as well as the staggered model. This difference stems from the inherent tradeoff between synchronization quality and convergence time.

The simultaneous startup experiments represent both the worst and expected case. That is, the effects of the race condition are most pronounced with simultaneous startup, such as after power failures and bulk management operations. The staggered set of experiments represents the best case because it perfectly prevents the race condition. However, under the same set of assumptions that form the basis of Bsync, it is very difficult to have an algorithm that simultaneously guarantees synchronization,

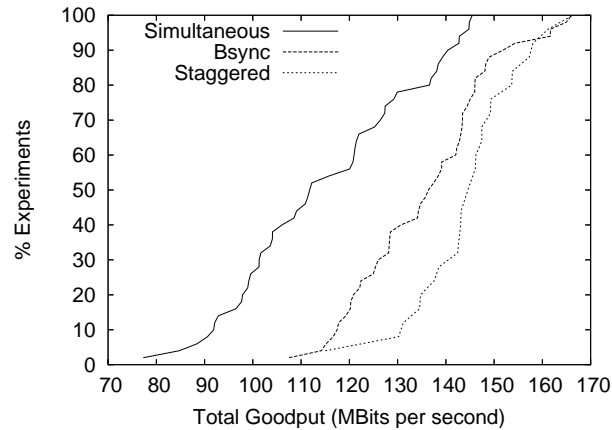


Figure 3.11: Performance of Bsync with the RSSI Max policy compared with simultaneous and staggered startups for 802.11a.

has a reasonable convergence time, and does not make any assumptions about the LAN connectivity of the APs.

Figure 3.12 shows the performance of Bsync RSSI Sum in an 802.11b environment compared to the Fixed, Random, and Least Occupied policies with a fully staggered start. Due to time constraints over the holidays there is not a complete data set that includes all combinations of policies and startup models. However, the staggered startup model is an upper bound for the Least Occupied policy. That is, no other synchronization technique can do better. Therefore it represents the best possible scenario for the policy. The fixed policy is included to provide a frame of reference to interpret the absolute performance numbers on the graph. The performance of random is independent of the startup model used. Bsync RSSI Sum outperforms the other policies, including Least Occupied with the most generous startup model, even though Bsync used the synchronized startup model.

3.8.2 Convergence Time

In addition to evaluating the goodput performance of a particular channel assignment, I also measured the converge time required to achieve that assignment.

Table 3.2 shows the number of seconds required for a given policy to converge

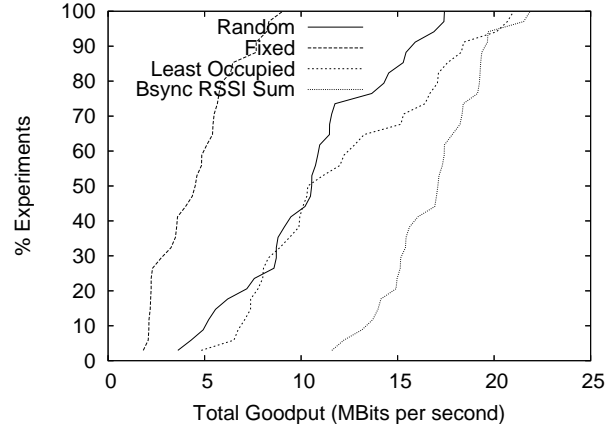


Figure 3.12: Performance comparison for Bsync with the RSSI Sum policy and a subset of the other 802.11b policies taken while the building was closed over the holidays. The Least Occupied policy employed a fully staggered startup.

Table 3.2: Convergence times for the Least Occupied policy across all startup models. Results are from 802.11a with a dwell time of 1 second. Each point is an average of 10 runs.

Policy	Startup Model	Convergence Time
Random	Simultaneous	8 sec
Least Occupied	Simultaneous	15 sec
Least Occupied	Bsync	55 sec
Least Occupied	Staggered	985 sec

to a channel assignment under different startup models. Bsync convergence time is 15 times faster than the staggered startup model, and only 4 times slower than the simultaneous model. As a result, I conclude that the convergence time of Bsync for a large, dense network is acceptable and practical.

As an upper bound, I can calculate the maximum Bsync convergence time by considering how long it spends in each phase. Referring back to Figure 3.9, which shows each phase labeled with its duration, let d be the dwell time for the scanning phase of the algorithm, and c be the number of available channels (which varies due to technology and geopolitical region). The total scan time, S , is then defined as $S = d * c$. The scan phase of Bsync takes S seconds, as does the scheduling and announcement phases.

In addition to these fixed values, there is a variable timeout applied for each AP in the selection schedule. This timeout value is initially set to two seconds for each AP; if there are 10 APs to pick before a given AP, for instance, then that AP would have an additional 20-second delay. To help maintain selection order, however, APs report their remaining time until selection in the beacon frames. Other APs that overhear this value use it to establish a more accurate timeout, however a strict bound of 2 seconds since the previous selection is maintained to ensure timely progress. As a result, the actual contribution of the timeouts to the total running time can not exceed $2 * n$ before any particular AP in the selection order, which is bounded by the total number of APs in the network (n). In all the experiments d was set to 1 second. The final upper time bound is:

$$T_u = 3 * d * c + 2 * n$$

3.9 Explaining the Goodput Differences

While the previous sections presented the overall goodput differences among the various channel assignment algorithms, they did not investigate the underlying cause. A number of interrelated mechanisms in 802.11 indirectly relate to the channel assignment and impact overall network goodput. Consider, for example, the automatic rate selection logic built into all 802.11 products. If one channel assignment caused an AP to operate at a suboptimal rate, then this assignment negatively impacts goodput.

To explore underlying causes of performance differences among channel assignment algorithms, I augmented the 802.11 madwifi driver to collect statistics related to link-level events: transmission rate, retries, losses, and contention. Some of this information is directly available from the wireless card itself, and I can infer the remainder from experimental observations.

802.11 has multiple rate encodings available for transmission. The lower speeds are more resilient to interference, while the higher ones more susceptible. This interference robustness tradeoff creates a challenge for the wireless radios. For immedi-

ate performance, a radio will want to transmit at the highest possible rate. A single loss, however, is potentially more harmful to overall network goodput than transmitting tens of frames at a slightly lower rate. As a result, wireless cards constantly reevaluate the effectiveness and loss characteristics of a particular rate, and adapt the rate they use to current wireless conditions.

I compute a transmission rate metric that tracks the performance potential lost due to transmission at a rate other than the fastest rate, regardless of whether or not the fastest rate would result in successful reception. More concretely, the metric computes the amount of airtime required to send the frame at the rate selected by the wireless card, computes the amount of time required to send the frame at the highest available rate, and subtracts the two values. The result is the amount of airtime unused by transmitting at the lower rate. This airtime is summed across the all packet transmissions in an experiment and represents the lost potential due to the use of a slower transmission rate. The maximum rate represents an upper bound and is useful for comparing different algorithms because all experiments were performed in the same environment with the same inherent rate limitations.

In addition to adjusting transmission rates, the 802.11 MAC protocol listens for radio silence before transmitting a frame. If one radio observes another radio's transmission, the listening radio delays sending its frame until such a time as the other node is no longer transmitting. This is the fundamental property behind the medium contention metric, which tracks the amount of time wasted because two radios were contending for the medium at the same time.

Since the wireless card does not provide a direct way to measure contention, I measure it indirectly. I compute the amount of time required to transmit a frame, including all MAC framing overhead, using the transmission rate selected by the card. I take measurements of the actual time required to transmit each individual frame, and the difference between the two times is the amount of airtime wasted due to contention.

While this metric is a measurement of wasted airtime, it is not a measurement of wasted potential. If the wireless card had no other frames waiting to be transmit-

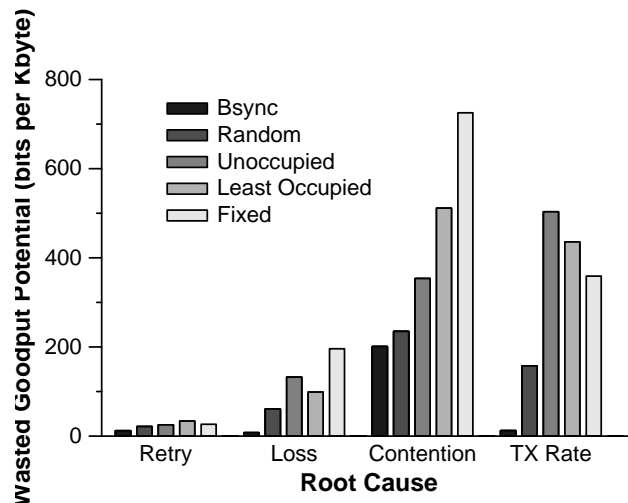


Figure 3.13: Performance lost potential due to underlying 802.11 properties as seen in one 802.11b experiment. The loss is measured in bits wasted per kilobyte successfully transmitted.

ted, then whatever delay was introduced by the contention would not impact the node's goodput. Consequently, I exclude measurement unless there is at least one frame queued up behind the frame that experienced contention.

I note that there are other factors ambiguously measured with this technique that bias the contention results. The 802.11 MAC layer retransmits unicast frames for which no acknowledgment is received from the intended wireless recipient. The timing technique used to measure contention includes time lost due to retransmissions, if any. The wireless card reports the number of retransmission attempts required to successfully send a frame; this information enables us to track lost potential due to retransmissions separately. Likewise, it is possible for a frame to fail entirely. When this occurs, the lost transmission time is credited to the loss metric.

The net result is three different metrics based on the same frame timing technique. I count *contention overhead* when the frame is successfully transmitted on the first attempt, *retry overhead* when the frame has one or more retries and is successful, and *loss overhead* when the transmission is not successful.

As a basis of comparison among various metrics, I convert all metrics originally measured in the time domain into bits. The conversion factor is based on the well-known 802.11 timing information. I use the amount of time required to transmit a single 1KB UDP packet, including all framing and other MAC overhead, at the highest available rate to determine the number of frames that could have been transmitted. The 1KB size is selected to match the frames used to measure goodput. Given that each frame is fixed in size, I compute the lost goodput by multiplying the number of lost frames by the size of the frame.

Figure 3.13 breaks down the cause of lost performance potential for a subset of the channel assignment policies in one of the experiments. It shows that Bsync produces the lowest loss in all categories among the channel assignment policies. Further, the transmit rate, retry, and loss categories all minimally contribute to lost performance potential, leaving contention as the primary factor.

Bsync was successful in creating a low-retry, low-loss environment that allows the transmission rate adaptation algorithm to select the near-maximum rate values for all frames. The results also indicate that further improvements beyond Bsync require reducing contention. In contrast, the other four policies lose performance potential to a combination of loss, transmit rate, and contention. The higher losses force the wireless card to use slower rates, as reflected in the transmission rate metric. Finally, the large contribution of the transmit rate to total lost potential underscores the need to provide a high quality radio environment that allows the fastest rate to be used.

3.10 Channel Selection Conclusions

Increasing density has made the careful management of shared spectrum a critical problem for 802.11 access networks. This chapter provides large-scale empirical measurements that directly relate 802.11 performance to the particular policies used by access points to select their initial channels. I have shown that startup synchronization can create selection races that significantly reduce performance and I described a

new distributed algorithm for efficiently minimizing the probability of such races. I have described empirical measurements demonstrating that the combination of this synchronization algorithm and a channel selection policy based on explicit channel measurements can provide superior performance to existing approaches. Finally, I have shown that the biggest reasons for these benefits are reductions in contention and increases in the average sending rate.

Chapter 4

802.11 Denial-of-Service

Vulnerabilities and Defenses

4.1 Introduction

The standardization committee responsible for developing 802.11 had the foresight to realize that wireless networks would be used to carry sensitive data. That is, data whose disclosure would create some form of irrevocable harm if seen by an adversary. To address this the specification provides the option of encrypting the network traffic before it is transmitted. Experience with actual 802.11 networks bears the importance of privacy out. For example, there are a number of point-of-sale devices that use wireless networks to process credit card transactions.

The designers were not the only people who realized the importance of privacy. Indeed, recent research has demonstrated basic flaws in 802.11's encryption mechanisms [FMS01, BGW01] and authentication protocols [ANJ01] – ultimately leading to the creation of a series of protocol extensions and replacements (e.g. WPA, 802.11i, 802.1X) to address these problems. However, most of this work has focused *primarily* on the requirements of access control and confidentiality, rather than availability.

While the protocol designers correctly anticipated the need to protect privacy, they failed to realize the need for availability. There are circumstances, like sensors

monitoring patients in a hospital or temperature monitors on a nuclear reactor, where a lack of data is unacceptable. Continuing with the hospital example, if a patient entered cardiac arrest but the heart rate monitor couldn't transmit that information to the nurse, the consequences could be severe. Unfortunately issues surrounding the availability of 802.11 networks are completely unaddressed in the standard. In fact, 802.11 is highly vulnerable to denial-of-service attacks, which prevent legitimate users from using the network. This chapter focuses on the problem of availability in 802.11 networks.

In general, denial-of-service attacks are an important, yet vexing, security problem. All networks, wired or wireless, are vulnerable to attacks on their availability of some form or another. Ultimately, what differentiates these vulnerabilities is how much effort they require to exploit and how much exposure risk the attacker must tolerate. For example, a determined attacker may physically disrupt a wired communications channel by severing a link, but this requires physical access to the network. More simply, an attacker may overwhelm the channel's capacity by sending thousands of spurious packets, but this same flood can provide a means for tracking an attack back to its origin [SWKA00, Sto00, BC00]. Perhaps the most insidious attacks leverage weaknesses in the underlying signaling, routing or media access protocols to prevent the channel from being used while making minimal demands on the attacker.

While all of these vulnerabilities are present in wired networks, they are particularly threatening in the wireless context. In particular, without a physical infrastructure, an attacker is afforded considerable flexibility in deciding where and when to attack, as well as enhanced anonymity due to the difficulty in locating the source of individual wireless transmissions. Moreover, the immaturity and limitations of today's wireless network management tools greatly decreases the probability that an attack on low-level 802.11 functionality will be detected. Finally, as I will show, if the attacker is clever, the channel may be disrupted using relatively infrequent transmissions, with low power consumption and minimal identification risk.

In this chapter I analyze the efficacy and practicality of attacks on the 802.11 MAC protocol. Through a combination of implementation and simulation I show that

an attacker can easily control the availability of existing 802.11 networks. Moreover, I argue that a significant subset of these attacks will not be addressed by planned security upgrades incorporated in the WPA or 802.11i standards. Finally, I describe minor modifications to 802.11 implementations that eliminate or mitigate these attacks and evaluate their effectiveness. The rest of this chapter is structured as follows: Section 4.2 describes related security research concerning 802.11 networks while Section 4.3 describes and categorizes 802.11 denial-of-service vulnerabilities. I analyze attacks on these vulnerabilities in Section 4.4 along with the effectiveness of potential defenses, and conclude in Section 4.5.

4.2 Related Work

A great deal of research has already been focused on 802.11 network security. The first part of the 802.11 specification [Soc99] to fall under scrutiny was the wired equivalency protocol (WEP). WEP was intended to provide the same level of privacy over wireless networks as is present on wired networks. It primarily relies on encrypting payload data and using challenge-response based shared secret authentication. In 2001, Fluhrer et. al. identified recurring weak keys in WEP, and showed how to use them to recover the secret key [FMS01]. Once the key is known, an attacker can both fully utilize network resources and monitor the traffic of other network nodes. In a recent paper, Stubblefield et al, demonstrate an implementation of this attack that was able to recover a 128-bit WEP key purely through passive monitoring [SIR02].

In addition, a number of attacks that don't require knowledge of the shared secret have been identified. Borisov et al demonstrated the possibility of modifying WEP protected frames [BGW01]. They also show how to inject new messages, how to spoof authentication frames, and how to recover the plain text from encrypted frames. All these attacks are mountable using commodity hardware, without knowledge of the WEP key. Problems beyond WEP shortcomings have also been identified. Arbaugh et al discuss the weakness of Lucent's proprietary "closed networks" mechanism, and the

limitations of the widely-used MAC address based access control [ANJ01].

As part of his thesis [Lou01] Daniel Lough develops and applies the VERDICT attack taxonomy to 802.11 wireless network and identifies a number of vulnerabilities, including the deauthentication attack presented in this chapter. He does not, however, present the details of the attack or develop defense techniques. A 2002 Black-Hat presentation [BL02] by Baird and Lynn provides details and demonstrates implementations of three 802.11 MAC layer DoS attacks, including the deauthentication attack, but does not propose any solutions as this work does.

To provide a long-term solution to these and other problems, the 802.11 working group has proposed the standard use of the 802.1X protocol [Soc01] in future versions of 802.11. However, even 802.1x has been shown to be vulnerable in the wireless context. A. Mishra and W. Arbaugh identify two such vulnerabilities which have been tested in operational settings: session hijacking and man in the middle attacks [AA02]. They also recommend modifications to 802.1X increase its effectiveness. A number of the denial-of-service attacks I consider, particularly those arising from known weaknesses in 802.11's unauthenticated management layer, are well-addressed by the introduction of 802.1X, although practically speaking I expect these attacks to remain viable for the foreseeable future.

Congestion-based MAC layer denial of service attacks have also been studied previously. Gupta et al [GKF02] looked specifically at 802.11 ad hoc networks and show that traditional wireline-based detection and prevention approaches don't work, and propose the use of MAC layer fairness to mitigate the problem. Kyasanur and Vaidya also look at congestion-based MAC DoS attacks, but from a general 802.11 prospective, not the purely ad hoc prospective [KV02]. They propose a straight forward method for detecting such attacks. In addition they propose and simulate a defense where uncompromised nodes cooperate to control the frame rate at the compromised node. Unlike these papers, I focus on attacks that can be staged with a minimum of effort by the attacker.

4.3 Vulnerabilities

The 802.11 MAC layer incorporates functionality uniquely designed to address problems specific to wireless networks. In particular, this includes the ability to discover networks, join and leave networks, and coordinate access to the radio medium. The vulnerabilities discussed in this section result directly from this additional functionality and can be broadly placed into two categories: identity and media-access control.

4.3.1 Identity Vulnerabilities

Identity vulnerabilities arise from the implicit trust 802.11 networks place in a speaker's source address. As is the case with wired Ethernet hosts, 802.11 nodes are identified at the MAC layer with globally unique 12 byte addresses. A field in the MAC frame holds both the senders and the receivers addresses, as reported by the sender of the frame. For "class one" frames, including most management and control messages, standard 802.11 networks do not include any mechanism for verifying the correctness of the self-reported identity. Consequently, an attacker may "spoof" other nodes and request various MAC-layer services on their behalf.

Exemplifying this class of vulnerability is the *deauthentication attack*. After an 802.11 client has selected an access point to use for communication, it must then authenticate itself, using one of two mechanisms: null key authentication and shared key authentication. In the context of this vulnerability, the distinction between these two mechanisms is unimportant – in the end the client and the access point come to agree that they are authenticated. However, part of the authentication framework is a message that clients and access points can use to explicitly request deauthentication from one another. Unfortunately, this message itself is not authenticated using any keying material in current 802.11 networks. Consequently, the attacker may spoof this message, either pretending to be the access point or the client, and directing it to the other party (see Figure 1). In response, the access point or client will exit the authenticated state and will refuse all further packets until authentication is reestablished. How long reestablishment

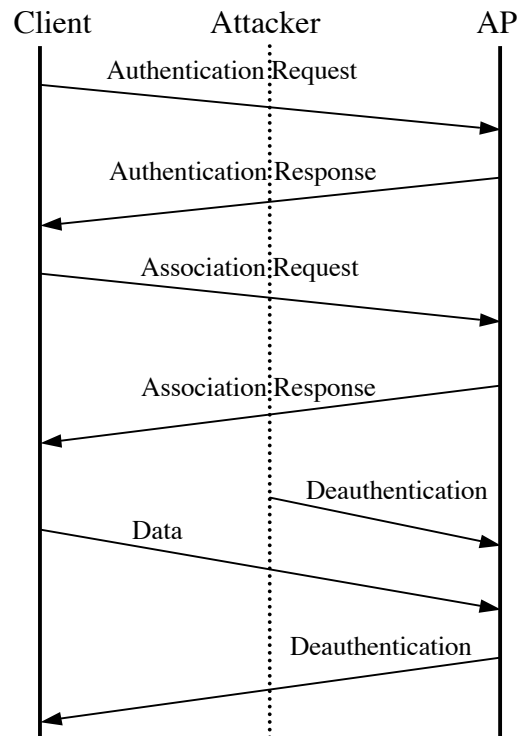


Figure 4.1: Graphical depiction of the deauthentication attack. It is noteworthy that the attacker needs only generate one packet for every six exchanged between the client and access point (AP).

takes is a function of how aggressively the client will attempt to reauthenticate and any higher-level timeouts or backoffs that may suppress the demand for communication. By repeating the attack persistently a client may be kept from transmitting or receiving data indefinitely.

One of the strengths of this attack is its great flexibility: an attacker may elect to deny access to individual clients, or even rate limit their access, in addition to simply denying service to the entire channel. However, accomplishing these goals efficiently requires the attacker to promiscuously monitor the channel and send deauthentication messages only when a new authentication has successfully taken place (indicated by the client's attempt to associate with the access point). As well, to prevent a client from "escaping" to a neighboring access point, the attacker must periodically scan all channels to

ensure that the client has not switched to another overlapping access point. Finally, it is not possible in current implementations to broadcast a deauthentication request and so the overhead of this method scales linearly with the number of clients to be attacked.

A very similar vulnerability may be found in the association protocol that follows authentication. Since a client may be authenticated with multiple access points at once, the 802.11 standard provides a special association message to allow the client and access point to agree which access point shall have responsibility for forwarding packets to and from the wired network on the client's behalf. As with authentication, association frames are unauthenticated, and 802.11 provides a disassociation message similar to the deauthentication message described earlier. Exploiting this vulnerability is functionally identical to the deauthentication attack. However, it is worth noting that the disassociation attack is slightly less efficient than the deauthentication attack. This is because deauthentication forces the victim node to do more work to return to the associated state than does disassociation, ultimately requiring less work on the part of the attacker.

There are many other vulnerabilities that take advantage of the implicit trust in MAC source addresses – some less well-known and others quite baroque.

For example, the power conservation functions of 802.11 have many identity-based vulnerabilities. To conserve energy, clients are allowed to enter a sleep state during which they are unable to transmit or receive. Before entering the sleep state the client announces its intention so the access point can start buffering any inbound traffic for the node. Occasionally the client awakens and polls the access point for any pending traffic. If there is any buffered data the access point delivers it and subsequently discards the contents of its buffer. By spoofing the polling message on behalf of the client, an attacker may cause the access point to discard the client's packets while it is asleep.

Along the same vein, it is possible to trick the client node into thinking there are no buffered packets at the access point when in fact there are. The presence of buffered packets is indicated in a periodically broadcast packet called the traffic indication map, or TIM. If the TIM message itself is spoofed, an attacker may convince a

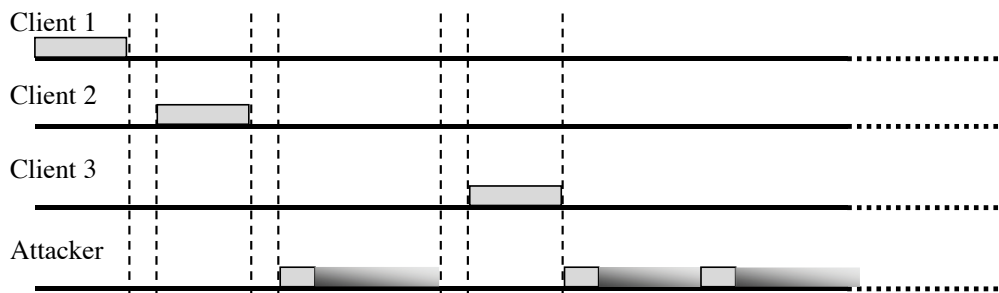


Figure 4.2: Diagram depicting the NAV Attack in action. The gradient portion of the attacker’s frame indicates bandwidth reserved by the NAV although no data is actually sent. Continually sending the attack frames back to back prevents other nodes from sending legitimate frames.

client that there is no pending data for it and the client will immediately revert back to the sleep state.

Finally, the power conservation mechanisms rely on time synchronization between the access point and its clients so clients know when to awake. Key synchronization information, such as the period of TIM packets and a timestamp broadcast by the access point, are sent unauthenticated and in the clear. By forging these management packets, an attacker can cause a client node to fall out of sync with the access point and fail to wake up at the appropriate times.

All of the vulnerabilities in this section can be resolved using 802.1X pre-authentication functionality and explicit authentication of management frames. However, given an installed base of over 15 million 802.11 devices and a significant managerial burden imposed by 802.1X (in particular, shared key management) it seems likely that these attacks will remain effective. In Section 4.4 I will demonstrate a simple system-level solution to these problems that can be implemented locally as a software upgrade – without any management burden and applicable in a public access environment.

4.3.2 Media Access Vulnerabilities

802.11 networks go through significant effort to avoid transmit collisions. Due to hidden terminals [BDSZ94] perfect collision detection is not possible and a combination of physical carrier-sense and virtual carrier-sense mechanisms are employed in tandem to control access to the channel. Both of these mechanisms may be exploited by an attacker.

First, to prioritize access to the radio medium four time windows are defined. For the purposes of this discussion only two are important: the *Short Interframe Space* (SIFS) and the longer *Distributed Coordination Function Interframe Space* (DIFS). Before *any* frame can be sent the sending radio must observe a quiet medium for one of the defined window periods. The SIFS window is used for frames sent as part of a preexisting frame exchange (for example, the explicit ACK frame sent in response to a previously transmitted data frame). The DIFS window is used for nodes wishing to initiate a new frame exchange. To avoid all nodes transmitting immediately after the DIFS expires, the time after the DIFS is subdivided into slots. Each transmitting node randomly and with equal probability picks a slot in which to start transmitting. If a collision does occur (indicated implicitly by the lack of an immediate acknowledgment), the the sender uses a random exponential backoff algorithm before retransmitting.

Since every transmitting node must wait at least an SIFS interval, if not longer, an attacker may completely monopolize the channel by sending a signal before the end of every SIFS period. While this attack is effective, it requires the attacker to expend considerable energy since on 802.11b networks an SIFS period is only 20 microseconds, leading to a duty cycle of 50,000 packets per second.

A more serious vulnerability arises from the virtual carrier-sense mechanism used to mitigate collisions from hidden terminals. Each 802.11 frame carries a *duration* field that indicates the number of microseconds that the channel is reserved. This value, in turn, is used to program the *Network Allocation Vector* (NAV) on each node. Only when a node's NAV reaches 0 is it allowed to transmit. This feature is principally used by the explicit *request to send* (RTS) / *clear to send* (CTS) handshake that can be used

to synchronize access to the channel when a hidden terminal may be interfering with transmissions.

During this handshake the sending node first sends a small RTS frame that includes a NAV large enough to complete the RTS/CTS sequence – including the CTS frame, the data frame, and the subsequent acknowledgment frame. The destination node replies to the RTS with a CTS, containing a new NAV updated to account for the time already elapsed during the sequence. After the CTS is sent, every node in radio range of either the sending or receiving node will have heard the NAV and will respect it for the duration of the future transaction. While the RTS/CTS feature is rarely enabled in practice, respecting the virtual-carrier sense function indicated by the NAV is mandatory in all 802.11 implementations.

An attacker may exploit this vulnerability by asserting a large NAV which prevents well-behaved clients from gaining access to the channel (as shown in Figure 2). While it is possible to use almost any frame type to assert the NAV, including an ACK, using the RTS has some advantages. Since a well-behaved node will always respond to RTS with a CTS, an attacker may co-opt legitimate nodes to propagate the attack further than it could on its own. Alternatively, this approach allows an attacker to transmit with extremely low power or using a directional antennae, thereby reducing the probability of being located. The maximum value for the NAV is 32677, or roughly 32 milliseconds, so an attacker need only transmit approximately 30 times a second to jam all access to the channel.

Finally, it is worth noting that RTS, CTS and ACK frames are not authenticated in any current or upcoming 802.11 standard. Moreover, even if they were authenticated, this would only provide non-repudiation since, by design, the virtual-carrier sense feature impacts all nodes on the same channel.

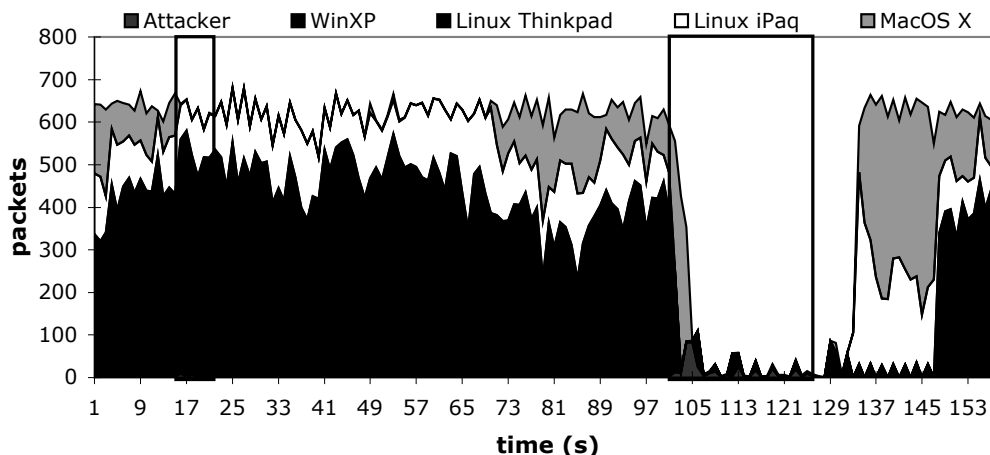


Figure 4.3: Packets sent by each of the 4 client nodes during the deauthentication attack. The first attack, against the MacOS client, started at second 15 and lasted 8 seconds. The second attack against all the clients started at 101 and lasted for 26 seconds. The attacking node consumes a negligible amount of bandwidth due to the rate limiting.

4.4 Practical Analysis of Attacks and Defenses

Attacks of the nature presented in this chapter typically have a sound theoretical background, but run into hard, unforeseeable problems in practice. With this in mind I attempted to implement a number of the different attacks. My efforts were met with varying degrees of success, and in some cases I resorted to simulation to make up for the limitations of commodity hardware, but in each case this provided insight into which attacks should be the most alarming.

From a practical perspective, one of the key questions is “what commodity hardware is capable of generating the frames required for the attack?”. I feel that implementation in commodity hardware is an important barrier, since it dramatically expands the set of potential attackers. At first glance this appears to be a trivial problem since all NICs are able to generate arbitrary bit patterns. However, in practice, all 802.11(a,b) NICs I am aware of implement key MAC functions in firmware and moderate access to the radio through a constrained interface. The implementation of this firmware, in turn, dictates which cards can be used effectively by an attacker. For example, not all NICs

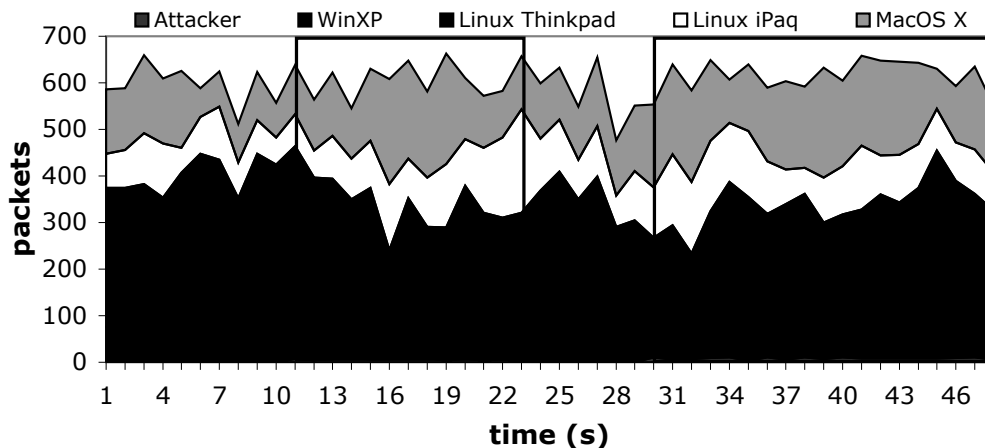


Figure 4.4: Packets sent by each of the 4 client nodes during the deauthentication attack with an access point modified to defend against this attack. The first attack, against the MacOS client, started at second 10 and lasted 12 seconds. The second attack against all the clients started at 30 and lasted through the end of the trace. The attacking node consumes a negligible amount of bandwidth due to the rate limiting.

will accept raw 802.11 frames for transmission, and even the most flexible firmware interfaces, such as that provided by the Atheros Chipset, still overwrite key protocol fields with computed values. Most NIC's firmware will silently filter packets using reserved or illegal values in the frame header fields or those containing control packets (such as RTS or CTS). Moreover, the firmware frequently overwrites key fields such as the duration field and the frame check sequence. In the end, I found that it was possible to exploit most of the identity-based vulnerabilities purely through software, but the media-access vulnerabilities could not be similarly exploited.

In the remainder of this section, I analyze my implementation of the deauthentication attack and a preliminary defense mechanism, followed by a similar examination of the NAV attack and defense driven by simulation.

4.4.1 Deauthentication Attack

My implementation of this attack promiscuously monitors all network activity, including non-data 802.11 frames, and matches the source and destination MAC

address against a list of attack targets. If a data or association response frame is received from a target, I issue a spoofed deauthentication frame to the access point on behalf of the client. To avoid buffer overflow in congested networks on the attacking machine, deauthentication frames are rate limited to 10 frames per second per client. This limit is reset when an access point acknowledges receipt of a deauthentication frame.

I tested this implementation in a small 802.11 network composed of 7 machines: 1 attacker, 1 access point, 1 monitoring station, and 4 legitimate clients. The access point was built using the Linux HostAP driver, which provides an in-kernel software-based access point. Each of the clients attempted to transfer, via ftp, a large file through the access point machine – a transfer which exceeded the testing period. I mounted two attacks on the network. The first, illustrated by the thin rectangle in Figure 3, was directed against a single client running MacOS X. This client's transfer was immediately halted, and even though the attack lasted less than ten seconds, the client did not resume transmitting at its previous rate for more than a minute. This amplification was due to a combination of an extended delay while the client probed for other access points and the exponential backoff being employed by the ftp server's TCP implementation.

The second attack, delineated by the wider rectangle in the same figure, was directed against all four clients. Service is virtually halted during this period, although the Windows XP client is able to send a number of packets successfully. This anomaly has two sources. First, these are not data packets from the ftp session but rather UDP packets used by Window's DCE RPC service and not subject to TCP's congestion control procedure. Second, there is a small race condition in my attack implementation between the time a client receives the successful association response and the time the attacker sends the deauthentication frame. The WinXP client used this small window to send approximately ten UDP packets before the attacking node shut them down. Modifying the implementation to send the deauthentication packets after both authentication and association would mitigate this effect.

A number of smaller, directed attacks were performed in addition to those in

figure 3. The small tests were done using the extended 802.11 infrastructure found at UCSD with varied victims. Recent versions of Windows, Linux, and the MacOS all gave up on the bad access point and kept trying to find others. Slightly older versions of the same systems never attempted to switch access points and were completely disconnected using the less sophisticated version of the attack. The attack even caused one device, an HP Jornada, to consistently crash.

The deauthentication vulnerability can be solved directly by explicitly authenticating management frames, as is planned using 802.1X in future 802.11 firmware. However, given the millions of existing 802.11 devices without this functionality, and the overhead in managing a shared key system, it seems likely that many 802.11 networks will be vulnerable to these attacks for some time. However, a system-level defense can be constructed against these same attacks by delaying the effects of deauthentication or disassociation requests. By queuing such requests for a specified period of time (e.g. 5-10 seconds) before acting, a node has the opportunity to observe subsequent packets from the client. If a data packet arrives after a deauthentication or disassociation request is queued, that request is discarded – since a legitimate host would never generate packets in that order. This approach has the advantage that it can be implemented with a simple firmware modification to existing NICs and access points, without requiring a new management structure.

To test this defense I modified the access point used in my experiments as described above, using a timeout value of 10 seconds for each management request. I then executed the previous experiment again using the “hardened” access point. The equivalent results can be seen in Figure 4. From this graph it is difficult to tell that the attack is active, and the client nodes continue their activity oblivious to the misdirection being sent to the access point.

My proposed solution to the deauthentication problem has a potential impact on nodes that roam between access points. For roaming to function correctly a mobile node disassociates from one access point and associates with another. The association determines how the packets destined for the mobile node are routed. In certain cir-

cumstances leaving the old association established for an additional period of time may prevent the routing updates necessary to deliver packets through the new access point. Or, in the case of an adversary, the association could be kept open indefinitely using spoofed packets. While both these situations are possible, I will show they are not likely to occur in practice.

There are two main infrastructure configurations that support roaming. For lack of a better name I refer to these as “intelligent” and “dumb”. In the “intelligent” configuration the access points have a means of coordination. This coordination can be used to, among other things, update routes for and transfer buffered packets between access points when a mobile node changes associations. Since the wireless standards don’t specify a coordination interface, they tend to be vendor specific and only work within a single vendor’s product line. In contrast “dumb” access points have no means of coordination. They rely on the network technology connecting the access points to handle routing changes when a node switches associations.

Intelligent infrastructures, due to their preexisting coordination, are easily modified to avoid the aforementioned problems caused by the deauthentication timeout. Since the mobile node must associate with the new access point before it can transmit data, and since the access points are coordinated (either directly or through a third party), the old access point can be informed when the mobile node makes a new association. Based on this information the old access point can immediately honor the clients deauthentication request. As an alternative to direct notification all packets for a host “in limbo” can be forwarded to an authoritative 3rd party who decides which access point transmits the data. Both of these solution leverage preexisting coordination channels.

Dumb infrastructures are a little more problematic because of their lack of coordination and reliance on the underlying network topology. If that underlying topology is a hub, which is a rarity these days, there is no problem because all packets are already delivered to all access points. If the underlying topology is routed, then either the access points must have some mechanism to coordinate routing changes and fall in the “intelli-

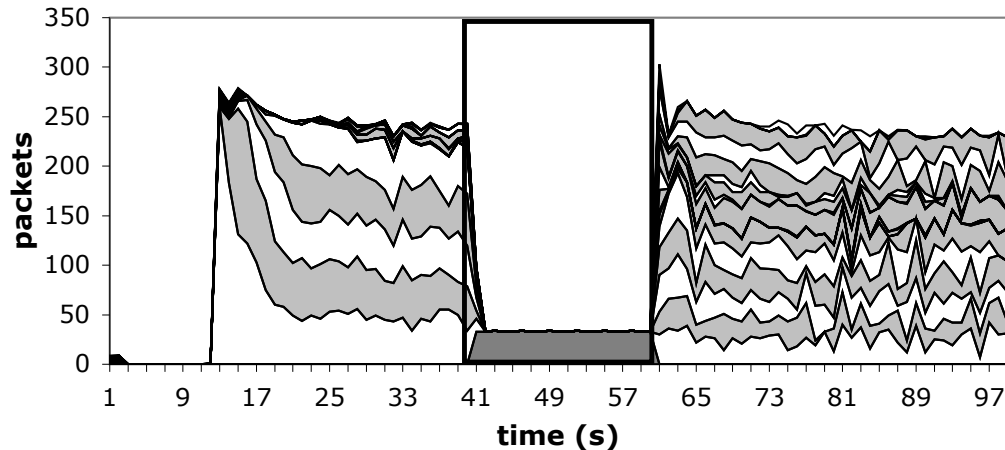


Figure 4.5: Results from the ACK based NAV Attack simulation with 18 client nodes. The attack begins at time 40 and ends at time 60. The dark region at the bottom of the graph during the attack is the attacker.

gent” category or the mobile client is issued a new transport layer address and no MAC conflicts are exposed. That leaves switched topologies. Switches learn the MAC addresses based on observed traffic. Existing switches already gracefully support moving a MAC from one port to another, but have problems when one MAC is present across multiple ports. In the non-adversarial case the mobile node will switch access points, proceed to send data using the new access point, and cease sending data through the old access point. From the switches perspective this is equivalent to a MAC switching ports. The mobile node may not receive data packets until it has sent one allowing the switch to learn its new port, but that limitation applies regardless of the deauthentication timeout. In the adversarial case the attacking node could generate spoofed traffic designed to confuse the switch. But even in this case there not an additional attack. If the access point was WEP enabled or otherwise protected it would not relay the packet and the switch wouldn’t get confused. If it wasn’t WEP enabled, or if the WEP key had been cracked, the attacker can mount this attack regardless of the deauthentication timeout, so there are no new vulnerabilities.

4.4.2 NAV attack

Motivated by the success of the previous attack, I attempted an implementation exploiting the NAV vulnerability. However, as stated earlier, I was unable to coerce any of my commodity NICs to generate packets with explicitly modified *duration* values. Instead, the remainder of this section explores the NAV vulnerability in the context of the popular ns simulator.

I implemented the NAV attack by modifying the ns [Sim] 802.11 MAC layer implementation to allow arbitrary NAV values to be sent periodically, 30 times a second, by the attacker. The attacker's frames were sent using the normal 802.11 access timing restrictions, which was necessary to prevent the attacker from excessively colliding with other in-flight frames (and thereby increase the amount of work required of the attacker). In addition the attacker was modified to ignore all NAV values transmitted from any other node. The network topology was chosen to mimic many existing 802.11 infrastructure deployments: a single access point node, through who all traffic was being sent, 18 static client nodes and 1 static attacker node, all within radio distance of the access point. As with the previous experiments, ftp was used to generate the long-lived network traffic. I simulated attacks using ACK frames with large NAV values, as well as the RTS/CTS sequence described earlier. Figure 5 shows the ACK flavor of the NAV Attack in action, but both provided similar results: the channel is completely blocked for the duration of the attack.

The NAV attack is much harder to defend against in practice than the deauthentication attack.

One approach to mitigate its effects is to place a limit on the NAV values accepted by nodes. Any packet containing a larger NAV value is simply truncated to the maximum value allowable. Strict adherence to the required use of the NAV feature indicates two different limits: a low cap and a high cap. The low cap has a value equal to the amount of time required to send an ACK frame, plus media access backoffs for that frame. The low cap is usable when the only packet that can follow the observed packet is an ACK or CTS. This includes RTS and all management (association, etc) frames.

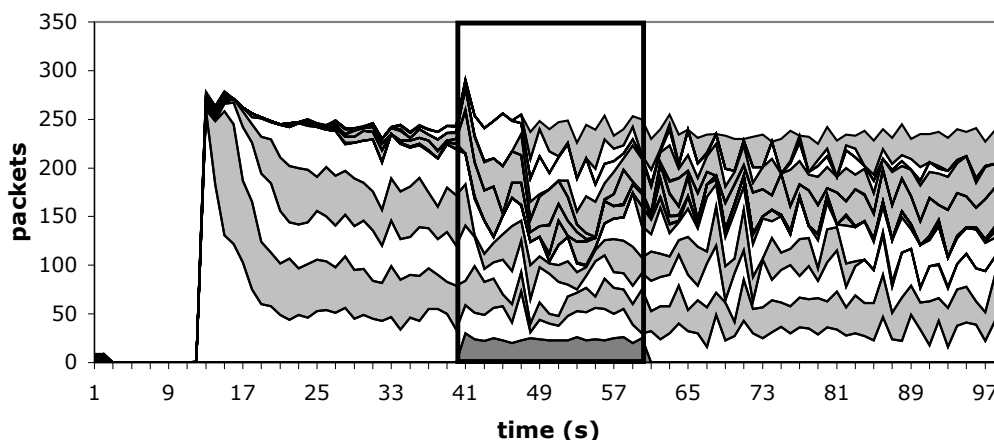


Figure 4.6: Results from the ACK based NAV Attack simulation with 18 client nodes modified to implement defense. The attack begins at time 40 and ends at time 60. The dark region at the bottom of the graph during the attack is the attacker.

The high cap, on the other hand, is used when it is valid for a data packet to follow the observed frame. The limit in this case needs to include the time required to send the largest data frame, plus the media access backoffs for that frame. The high cap must be used in two places: when observing an ACK (because the ACK may be part of a MAC level fragmented packet) and when observing a CTS.

I modified my simulation to add these limits, assuming that a value of 1500 bytes as the largest packet. While this isn't strictly the largest packet that can be sent in an 802.11 network, it is the largest packet sent in practice because most packets are bridge to ethernet, which has a roughly 1500 byte MTU. Figure 6 shows a simulation of this defense under the same conditions as the prior simulation. While there is still significant perturbation, many of the individual sessions are able to make successful forward progress. However, I found that simply by increasing the attacker's frequency to 90 packets per second, the network could still be shut down. This occurs because the attacker is using ACK frames, whose NAV is limited by the high cap.

To further improve upon this result requires us to abandon portions of the standard 802.11 MAC functionality. At issue is the inherent trust that nodes place in the NAV value sent by other nodes. By considering the different frame types that carry

NAV values I can define a new interpretation of the NAV that allows us to avoid most possible DoS attacks. The four key frame types that contain NAV values are ACK, data, RTS, and CTS, and I consider each in turn.

Under normal circumstances the only time a ACK frame should carry a large NAV value is when the ACK is part of a fragmented packet sequence. In this case the ACK is reserving the medium for the next fragment. If fragmentation is not used then there is no reason to respect the NAV value contained in ACK frames. Since fragmentation is rarely used (largely due to the fact that default fragmentation thresholds significantly exceed the Ethernet MTU) removing it from operation altogether will have minimal impact on existing networks.

Like the ACK frame, the only legitimate occasion a data frame can carry a large NAV value is if it is a subframe in a fragmented packet exchange. Since I have removed fragmentation from the network, I can safely ignore the NAV values in all data frames.

The third frame type to be concerned with is the RTS frame. The RTS frame is only valid in an RTS-CTS-data transmission sequence. If an RTS is seen on the network, it follows that the node seeing the RTS will also be able to observe the data frame. The 802.11 specification precisely defines the time a CTS frame, and subsequent data frame, will be sent. Therefore the NAV value in the RTS packet can be treated speculatively – respected until such time as a data frame should be sent. If the data frame is not observed at the correct time, either the sender has moved out of range or the RTS request was spoofed. In either case it is safe for the other node to ignore the duration of the NAV.

The last frame to consider is the CTS frame. If a lone CTS frame is observed there are two possibilities: the CTS frame was unsolicited or the observing node is a hidden terminal. These are the *only* two cases possible, since if the observing node wasn't a hidden terminal it would have heard the original RTS frame and it would be handled accordingly. If the unsolicited CTS is addressed to a valid, in-range node, then only the valid node knows the CTS is bogus. It can prevent an this attack by responding

to an such a CTS with a null function packet containing a zero NAV value – effectively undoing the attackers channel reservation. However, if an unsolicited CTS is addressed to a nonexistent node, or a node out of radio range, this is indistinguishable from a legitimate hidden terminal. In this case, there is insufficient information for a legitimate node to act. The node issuing the CTS could be an attacker, or they may simply be responding to a legitimate RTS request that is beyond the radio range of the observer.

An imperfect approach to this final situation, is to allow each node to independently choose to ignore lone CTS packets as the fraction of time stalled on such requests increases. Since hidden terminals are a not a significant efficiency problem in most networks (as evidenced by the fact that RTS/CTS are rarely employed) setting this threshold at 30 percent, will provide normal operation in most legitimate environments, but will prevent an attacker from claiming more than a third of the bandwidth using this attack.

It should also be noted that existing 802.11 implementations use different receive and carrier sense thresholds. The different values are such that, in an open area, the interference radius of a node is approximately double its transmit radius. In the hidden terminal case this means that although the hidden terminal can not receive the data being transmitted, it still detects a busy medium and won't generate any traffic that would interfere with the data, so the possibility of an unsolicited CTS followed by an undetectable data packet is very low.

But ultimately the only foolproof solution to this problem is to extend explicit authentication to 802.11 control packets. Each client-generated CTS packet contains an implicit claim that it was sent in response to a legitimate RTS generated by an access point. However, to prove this claim, the CTS frame must contain a fresh and cryptographically signed copy of the originating RTS. If every client shares keying material with all surrounding access points it is then possible to authenticate lone CTS requests directly. However, such a modification is a significant alternation to the existing 802.11 standard, and it is unclear if it offers sufficient benefits relative to its costs. In the meantime, the system-level defenses I have described provide reasonable degrees of protec-

tion with extremely low implementation overhead and no management burden. Should media-access based denial-of-service attacks become prevalent, these solutions could be deployed quickly with little effort.

4.5 Conclusion

802.11-based networks have seen widespread deployment across many fields, mainly due to the physical conveniences of radio-based communication. This deployment, however, was predicated in part on the user expectation of confidentiality and availability. This chapter addressed the availability aspect of that equation. I examined the 802.11 MAC layer and identified a number of vulnerabilities that could be exploited to deny service to legitimate users. Two such attacks were explored in detail, including testing in experimental environments and simulation. In each case the attack was shown potent.

In addition to demonstrating the attacks, I provided and analyzed countermeasures that are practical to implement on existing consumer hardware. In one instance the countermeasures completely defeated the attack, while in the other they severely limited its effectiveness. This chapter helps to underscore the care that must be taken when deploying 802.11 networks in mission critical applications.

4.6 Special Thanks

A special thanks goes out to the residents of csl-south, who were at times unwitting victims of the DoS exploits.

This chapter, in full, is a reprint or has been submitted for publication of the material as it appears in the Proceedings of USENIX Security Symposium 2003, Bellardo, John; Savage, Stefan. The dissertation author was the primary investigator and single author of this paper.

Chapter 5

Future Directions and Insights

There are a number of future directions that build on this work. They fall into two categories, work that is directly related to that presented here, and work that is not directly related, but has the same goals and motivations. This chapter reviews each category in turn.

Chapter 4 describes two specific denial-of-service attacks vulnerabilities in the 802.11 specification, along with techniques that protect against them. These security problems are just the tip of the iceberg. In general there is a need for a range of similar defenses to improve the robustness of the protocol. Some of these defenses are merely adaptations of the techniques in chapter 4, while others require additional originality.

Unfortunately there is an inherent security limitation in wireless networks – the shared wireless medium. Being forced to use a common resource requires a certain level of cooperation among devices in the network. A fruitful avenue of research is to determine just what that minimum level of cooperation is to achieve an operational network. Of course there is probably not a simple answer, but instead a spectrum of options where increasing cooperation decreases security while simultaneously increase network performance or utilization. Gaining a good understanding of this tradeoff space would greatly benefit the development and deployment decisions of future wireless protocols.

The area of wireless channel selection needs some clarification and direction. As it stands, there have been a number of proposed algorithms that all aim to address

similar problems. Typically the only factor differentiating the motivations of all these algorithms is the set of assumptions around which the algorithm is based. To the best of my knowledge, this work is the first to build and utilize a large wireless testbed to evaluate channel selection algorithms. It would be rather beneficial to use the infrastructure and run a series of head to head comparisons between all the currently proposed algorithms. In addition to the raw performance comparison, it can be reasonably expected to obtain insights into the various shortcomings in the existing algorithms, and hence guide future research efforts.

The overarching goal behind this thesis, to address shortcomings in the 802.11 specification, is a formidable task, and, in general, there is a substantial amount of work left to do. One glaring hole is a lack of understanding the important interactions present in deployed networks. While there has been some progress made in this direction [CBB⁺06], it is far from complete.

In addition to better understanding of operational wireless networks in general, there are some parts of the 802.11 specification that are not being used for various reasons. One of these is the *point controller function* (PCF). Without going into the details of PCF, it should be possible to design and carry out a series of experiments that demonstrate the merits, or lack thereof, of PCF.

Chapter 6

Conclusion

Recent years have seen an explosion in growth of 802.11-based products. This happened in part because of the convenience and price point of the hardware. Unfortunately this same growth that is beneficial for the wireless market exposes shortcomings in the standard. This thesis addressed two such areas, security and management.

In particular I improved 802.11's robustness against denial-of-service attacks. I performed a security evaluation of the protocol, identified numerous vulnerabilities, and then tested and evaluated two of them using actual hardware. My experience during development provided insights into workable defense mechanisms, which I also tested and showed to work as conceived.

The second problem I addressed is the area of automatically managing large wireless networks, specifically looking at the problem of assigning channels to individual access points. I deployed a large wireless testbed at UCSD which was then used to measure a number of different selection algorithms. Based on these measurements I concluded that the RSSI Sum policy was the best. Additionally I identified a race condition common among scanning-based channel assignment algorithms. I proposed and evaluated a new coordination protocol, Bsync, that succeeds in avoiding the majority of the aforementioned race condition.

Appendix A

Wireless Testbed Manual

A.1 Chapter Outline

Large scale testbeds are neither easy or straightforward, and typically involve hundreds of man-hours worth of effort to design, implement, and deploy. The hallmark of my research is experimental understanding of real systems, and as such I've been involved in a number of infrastructure rollouts.

This chapter discusses the largest such rollout – a 180 radio wireless testbed deployed in the Computer Science and Engineering Building (EBU3B) at the University of California, San Diego (UCSD). This information serves two main purposes. First, it provides a rough blueprint for, and conveys information about the scope of, similar projects that other academic and industrial institutions may desire to undertake. Secondly it provides detailed operating information for the existing testbed with the intent of streamlining future research that leverages the testbed's unique capabilities.

A.2 Overview

Size is one of the key attributes of our wireless testbed, in two senses. Our testbed contains a large number of radios, 180, spread over a large indoor space, 150,000 square feet. Figure A.1 shows the deployment locations for four of the five floors covered in the Computer Science and Engineering building at UCSD.

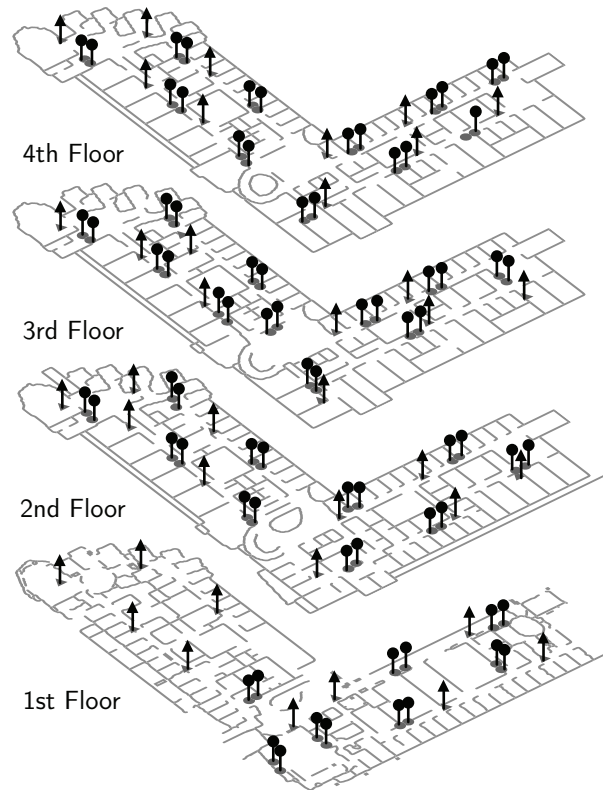


Figure A.1: CSE Building floorplan with wireless node locations depicted. This building comprises roughly 150,000 square feet spread over four floors (and a smaller basement, not shown). Circles indicate testbed nodes, and triangles indicate campus production access points. The basement houses an additional 12 nodes not shown in this figure.

Each node in our testbed contains two independent radios. This has a number of benefits. It enables research into multi-hop topologies, it enables measurement-based research projects that monitor two channels at the same point in space simultaneously, and it reduces the overall hardware costs of our infrastructure. It does, however, also have a drawback. Simultaneous transmissions from both radios are more likely to interfere with each other due to the close proximity, and the average distance between nodes is greater.

In addition to placing two radios in every node, I installed nodes in pairs (referred to as pods). Before installation I performed a series of bandwidth measurements to determine the minimum spacing necessary between two nodes in a pod without undue intra-pod interference. The nodes are installed as close to this minimum as conditions permit. This pairing provides passive measurement research projects the ability to monitor up to four different channels from what is effectively a single point in space. This is enough coverage to monitor the traditionally non-overlapping channels in 802.11b plus one channel in between.

Each node is directly connected to a dedicated wired network, which in turn connects to the building's network. The nodes are powered using Power over Ethernet. This reduces the installation complexity, expense, and improves the visual aesthetics.

In order to support the widest array of research, each node is an embedded computer that uses a Pentium class CPU and has 64 megabytes of non-volatile FLASH memory and 128 megabytes of RAM soldered on the logic board. The next section, *Node Specific Information* goes into detail regarding the hardware and software specifics for each node.

A.3 Node Specific Information

The nodes are the main component of the testbed. Getting them operational required a number of specific hardware and software settings, and in some case changes. This section details what is required for the testbed nodes.

A.3.1 Hardware Specifications

The wireless testbed is composed of two similar nodes, both manufactured by Soekris [Soea]: the net4826 [Soec] and the net4801 [Soeb]. Both boards use a 266 Mhz Geode SC1100 processor from AMD and have 128 megs of RAM. The 4826 has 64 megabytes of non-volatile FLASH memory soldered on the board while the 4801 sports a standard compact flash slot. The 4801 lacks PoE. As a result the 4826's are permanently deployed in building's hallways powered via PoE, while the 4801 are deployed in various labs where AC outlets are readily available.

Both boards have two mini-PCI slots, each of which is populated with an Atheros 5212-based 802.11 card. Atheros-based cards were chosen primarily because they employ a software-based radio. Software radios push the burden of protocol implementation out to the driver, which in turn provides a richer environment for performing wireless experiments.

A.3.2 Power-over-Ethernet customization

The Power-over-Ethernet (PoE) implementation on the net4826 is not fully 802.3af compliant. This contradicts the product specifications page and other marketing literature available from Soekris. The non-compliance was confirmed via email with Soekris's lead engineer.

802.3af, the PoE specification, provides two ways to deliver power to compliant devices. Option 1 is placing the voltage on the unused data pairs of the Cat-5 cable. The unused pairs are numbered 3 and 4. Pair 3 is composed of pins 4 and 5, while pair 4 is composed of pins 7 and 8.

The second option is delivering the power on the two *used* data pairs. These are pairs 1 and 2 composed of pins 1+2, and 3+6 respectively. The circuitry for accepting the power varies depending on which pairs the power is delivered on.

The 802.3af specification mandates that devices accept *both* power delivery options, however, as designed, the net4826 only accepts power on the unused pair. Unfortunately the PoE switches used in the testbed deliver power on the data pairs, and as

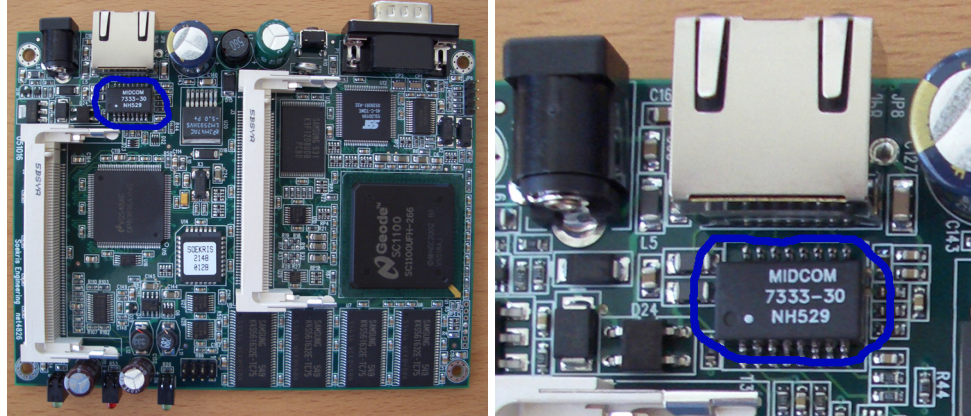


Figure A.2: The location of the ethernet isolation transformer on the Soekris net4826 board. The image on the left shows the whole board, while the image on the right is a close up shot of the transformer. In both images the transformer is circled.

such are incompatible with the 4826's.

Fortunately there is a way to modify the 4826's to accept power on the data pairs, but it does require a steady hand with a soldering iron. Accepting the power over the data pairs requires an ethernet isolation transformer with a center tap. The power is extracted using the center taps to avoid degrading the data signal past the point of usefulness. The ethernet isolation transformer used in the 4826, a MIDCOM 7090-37 [Mid], includes pins for the center taps. The PoE modification solders wires to those two pins, and loops them underneath the logic board to connect with the power input circuitry. Detailed soldering instructions are presented in the next paragraph.

The first step is to locate the ethernet isolation transformer IC. It is immediately behind the ethernet jack as indicated in figure A.2. Once located identify the two center tap pins. They are pins 2 and 6 of the IC as depicted in figure A.3. It may also be useful to refer to the data sheet for the IC, available at [Mid].

Once the pins are located solder one wire to each pin, being careful not to short other pins on the chip. This may require multiple attempts so keep some solder wick handy. Figure A.4 shows the IC with wires soldered to the pins.

Loop the wires around the side of the ethernet jack to the underside of the



Figure A.3: The location of the center tap pins for the ethernet isolation transformer on the Soekris net4826 board. Each of the two pins is circled.

board. Instead of attempting to locate a more appropriate location I simply soldered the other end of the wires to the unused pins in the ethernet jack. This has the effect of taking the power from the data pairs and placing it on the unused pairs, which is where the board expects the power to be delivered. It also has the effect of transmitting the power back through the unused pairs to whatever device, most likely a switch of some sort, the 4826 is connected to. Depending on the device this may or may not cause damage. It is of no consequence for our HP switches [Hew].

Use figure A.5 to locate the soldering locations for the other end of the wires. Note this is on the bottom of the logic board. Figure A.6 shows the wires soldered in the correct location.

That is it. Reassemble the case the enjoy. In case you are concerned, there is still adequate clearance for the logic board despite the newly added wires and their routing around the edge of the board.

A.3.3 Mounting Considerations

The net4826's are mounted using the standard enclosure as sold by Soekris, however the case does not come with a mounting solution. To get around that limitation I found a metal picture hanger at Home Depot [Hom] that has a security clip to prevent

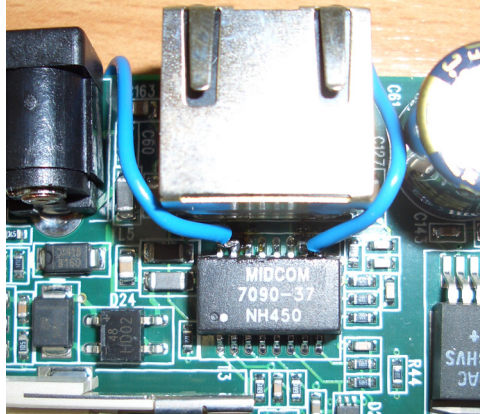


Figure A.4: A wire soldered to each of the center tap pins.

casual theft.

The bracket comes in two parts. The first part is screwed to the wall using drywall anchors or other suitable attachments. It is important that the bracket be electrically isolated from the screws. I experienced grounding problems because the metal bracket was in contact with the metal screw, which was in contact with a metal part of the building's structure. This resulted in difficult to diagnose power sharing problems where two APs would share the power provided by one PoE connection.

The second part of the bracket needs to be welded to the net4826's enclosure. Fortunately there is a machine shop on campus that does this work for a nominal fee. The name of the group is the Campus Research Machine Shop (CRMS), and it is located in the basement of Urey hall. TritonLink has a department listing page with a link to the home page for CRMS, which has all the contact information necessary to get a job started. It is necessary to remove the logic boards from all the cases, and obviously provide the brackets to the shop. It has typically taken between 2 and 4 days for the work to be completed, even on large orders.

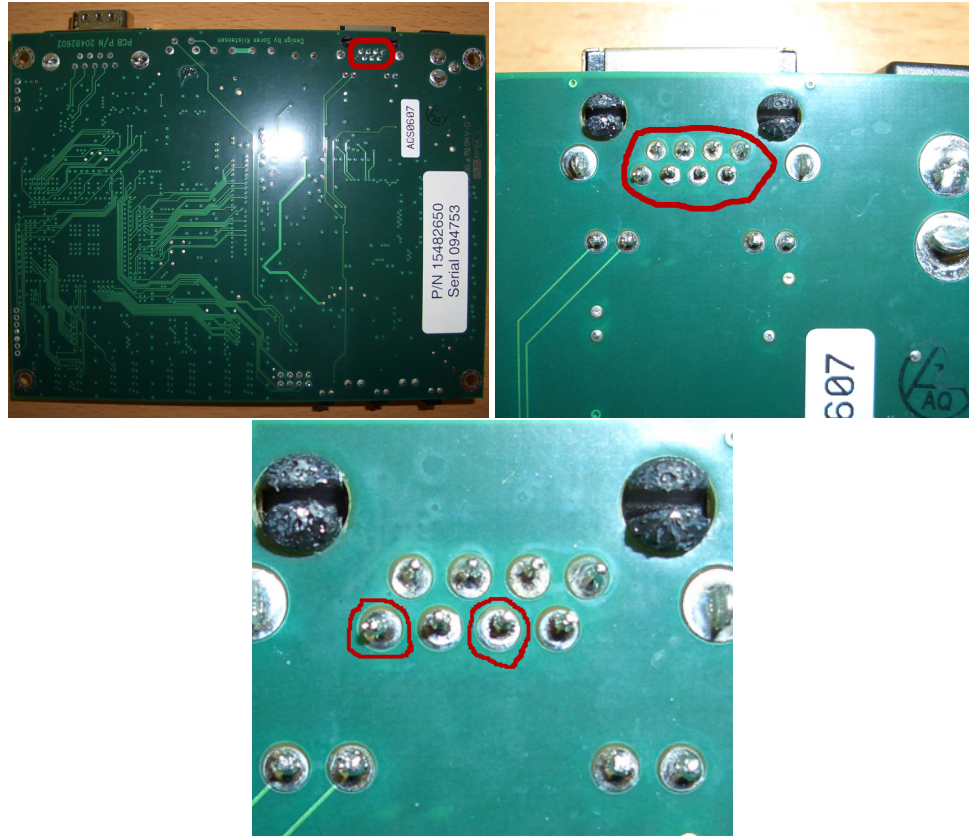


Figure A.5: The location of the ethernet jack pins on the bottom of the Soekris net4826 board. The image on the top left shows the whole board, the image on the top right is a close up shot of the ethernet jack pins, and the image on the bottom identifies the particular pins in question. The interesting parts of all images are circled.

A.4 Node Software

Each testbed node runs a version of linux that has been tailored for their unique characteristics. The base distribution is pebble, but then packages are added and removed. Complete custom applications are installed to streamline management. This section reviews the software configuration in depth.



Figure A.6: The wires soldered to the ethernet jack pins on the bottom of the Soekris net4826 board.

A.4.1 Pebble Linux Distribution

The base distribution used for the APs is Pebble [NYCc] available from NYC Wireless [NYCb]. Pebble was chosen because it is well suited to run off of compact flash. During normal operation the root filesystem is mounted read-only. This substantially reduces the wear and tear on the flash itself, increasing the lifetime of the node. For the few applications that need to write files, the distribution creates a small ramdisk mounted on /rw. This is initially populated with the files from /ro, so any changes that need to be persistent in the /rw filesystem across reboots need to be written in /ro. The distribution provides two convenience functions to write to flash, *remountrw* and *remountro*. *Remountrw* mounts the root filesystem writable so all changes are persistent. Once you are finished making changes, the *remountro* command remounts the system read-only, its normal operating state. For more operational details on pebble refer to the readme included in the distribution [NYCa]. At the time of this writing the version employed was v36. However the base distribution was aggressively modified to fit under the 64 megabyte flash limit in the 4826s.

In addition to removing unneeded files I also replaced the linux kernel in the distribution with a custom 2.4.31 kernel. The new kernel is patched to better take advantage of hardware features on the net4826, including the watchdog circuit. Another

kernel patch brings the wireless extension support up to version 26. I also reduced the drivers to the minimum set necessary to run the hardware, again to save space on the resource constrained device. Of course, a new kernel necessitates compiling all the corresponding kernel modules, so those are also updated.

The fully patch kernel source is available in the wireless project at `/projects/wireless-aps/soekris/pebble/pebble-v36/kernel-2.4.31-ucsd-soekris/linux-2.4.31-ucsd-soekris`.

A.4.2 Installation Process

The installation process is completely custom, due to the fact the flash is not removable from the net4826. At a high level the box performs a network boot into an install image, reformats and installs the pebble image and additional required packages, and then reboots. However understanding the process in enough detail to reproduce or modify it requires a more detailed explanation.

Network Booting

In order to network boot the BIOS is configured to attempt a PXE first, and boot from the local disk second. The information necessary for a network boot is delivered in three stages. First the box obtains an IP address using DHCP. Along with the IP address is a “next-server” record that instructs the PXE network boot loader to contact a tftp server for the next boot loader in the chain. PXE obliges and loads PXELinux, which in turn searches for a configuration files using a well-known progression of truncated MAC addresses.

Once an appropriately named configuration file is located PXELinux loads it and presents the boot options via the serial console. For the installation phase the default boot option is a NFS root boot into the installation image. After the option prompt times out it proceeds to boot into the NFS root image. It is worth noting that there are a number of kernel compile time options required to support NFS booting. Refer to the kernel `.config` file in the `linux-2.4.31-ucsd-soekris` directory for the exact options.

The NFS root image is another customized pebble image. There is a substantial security problem with NFS root. The two machines have to have mutual trust because the client needs write permission to the root filesystem. This also usually precludes booting a number of nodes from the same root image. However, pebble's read-only nature solves both of these problems. The NFS root image is made read-only, and pebble operates correctly because it already assumes a read-only root partition, abet flash instead of NFS. In our system the group's file server, homebrew, directly serves the NFS root image to all the testbed nodes. This installation-only root filesystem is available for inspection and modification at `/projects/wireless-aps/master-image/bootstrap`.

Installation Script Actions

After the usual startup scripts have run and the box has booted to the NFS root image, the custom installation script is run. This script is accessible at `/projects/wireless-aps/master-image/bootstrap/pebble-inst/install.sh`. The first action it takes is reformatting the flash and selecting the custom kernel image. Kernel selection is hard coded in the install script. Once the flash is reformatted the installation process untars the base pebble image to the flash. The base image is found in `/projects/wireless-aps/master-image/bootstrap/pebble-inst/pebble.v36.tar`. After this completes the `clean_packages.sh` script from the skeleton directory (see section A.4.3 for more information on that directory) is run to remove unused packages and install new ones. The current `clean_packages.sh` script removes the contents of a number of cache directories, man pages, and so on, and then installs the packages listed in section A.4.4.

Once the cleaning script has been executed all files from the skeleton directory are copied to the flash, potentially overwriting files that were also included in the base pebble distribution. This is the primary method to change configuration files and install unpackaged applications in the main testbed image. In addition to normal files there is a tar archive that contains special device files. This is necessary to preserve the major and minor device numbers associated with the device files. At this point ownership of files

in `/ro/home` is changed to match the user's account name instead of `root`.

The next step is to create a `lilo.conf` file. The script creates an entry for each kernel found in the newly installed filesystem, and an entry to perform a network boot reinstall. The set of kernel parameters required for the boxes is hardcoded into the installation script. After creating the `lilo.conf` file it runs `lilo` to install the bootloader on the flash.

After installing the bootloader the installation script looks for files specific to the particular machine being installed, keyed by IP address, and copies them over to the local flash image. This mechanism provides machine dependent configuration files which can be used for ensuring ssh host key consistency across installations, among other things.

After coping the machine specific files two configuration files are modified to reflect the IP and hostname of the box. Then the module dependency information is recomputed, the flash unmounted, and the box rebooted.

It is worth noting that, unless the default network boot action of reinstall is changed to local boot, the reinstallation process is recursive and never ending. Refer to the management tools in section A.6.5 for information on changing the default network boot option.

A.4.3 Pebble Skeleton Directory

The pebble skeleton directory contains all the custom machine generic and specific files that are installed during the installation process. The machine specific files are located in `/projects/wireless-aps/master-image/bootstrap/pebble-inst/mach-files` and keyed by IP address. The machine agnostic files are located in `/projects/wireless-aps/master-image/bootstrap/pebble-inst/skel`, however they should not be edited directly. Instead there is a process for updating a shadow directory and a script to copy the contents of the shadow directory into the actual skeleton directory.

The shadow directory is located in `/projects/wireless-aps/soekris/pebble/pebble-v36/skel`. Any programs that are compiled by hand are

pointed to this directory as the installation destination. This extra level of indirection allows for removal of unnecessary files to remain under the 64 megabyte limit. For example man pages are not needed on the nodes themselves. In fact the man application is not even installed.

All configuration files are also edited in the shadow skeleton directory. Once the shadow skeleton directory is satisfactory the files are copied to the real skeleton directory using the makefile in /projects/wireless-aps/soekris/pebble/pebble-v36. Make must be run as root to ensure appropriate file ownership is retained.

A.4.4 Additional Pre-Built Packages

Six packages are installed in addition to the packages included with the base pebble distribution. These are intentionally installed *without* resolving dependencies because doing so results in an image that doesn't fit within 64 megabytes. The packages are:

1. screen – Enables log-lived jobs that are started from an interactive shell while allowing the original connection to be terminated. Very useful for managing experiments manually.
2. sudo – To avoid passing out the root password to anyone and everyone who wants to do something interesting with the testbed.
3. libc6 – To enable dynamic linking of programs compiled using slightly different installations of pebble.
4. libglib1.2 – Same justification as libc6.
5. libstdc++5 – Enable dynamically linked c++ programs to run on the testbed.
6. less – Limited feature set of the “more” pager is overly restrictive.

A.4.5 Additional Custom-Built Packages

A number of application were compiled and installed from source, in addition to those pre-packaged applications discussed in section A.4.4. The source code for all the packages in this section can be found in `/projects/wireless-aps/soekris/pebble/pebble-v36/apps`.

Ganglia [The] was installed to provide near realtime operational status about the testbed nodes. Ganglia includes a monitoring component that reports a number of predetermined metrics, such as uptime and network usage, via local area multicast to an aggregation node. This data is presented via a web page that allows visualization and troubleshooting.

Hdparam [Gaz] is added to the distribution because the net4801's have the capability to hold a laptop hard drive, and at one point in the evolution of testbed each node actually had a disk drive. The utility allows tweaking of performance characteristics for the drive controller. Of interest in the testbed was the setting to enable DMA for the controller. DMA is disabled by default which wastes the box's precious CPU cycles.

In addition to monitoring via ganalia, monitoring is also available via SNMP [CFSD]. To achieve this I installed net-snmp [Teaa]. However, before installing it I added a substantial number of values to the MIB. These were all aimed at reporting detailed statistics about each wireless interface, and included such items as the number of retries, the number of frames sent at each available transmission rate, and so on.

Openslp [Teab] is used to aid metric collection via the SNMP reporting mechanism. Openslp is an implementation of the IEEE standards track service location protocol, based on Apple's Bonjour [AC]. It allows a monitoring node to query the network, via a broadcast probe, and discover all testbed nodes that are powered on. This enables monitoring software to dynamically and intelligently adjust their SNMP data collection as nodes join and leave the testbed.

Lastly I compiled and installed version 26 of the linux wireless tools [Tou]. These user-level tools are necessary to match the v26 wireless extensions that were added to the custom linux kernel.

A.4.6 Custom-Written Applications

There are two custom written applications installed on the testbed nodes. The first is daemon-watch, which is available in /projects/wireless-aps/daemon-watch. Daemon-watch forks a child process and continually relaunches it when it dies. The intent of this program is to provide a level of robustness to experimental processes which crash for unknown reasons.

The second application was not written from scratch, but rather heavily customized. It is a capture portal based on NoCatSplash [Wir] to provide roaming authentication for testbed users, insuring they know they are using an experimental platform. Since the customizations are highly involved there is a whole section, A.4.11, devoted to them in this chapter.

A.4.7 BIOS Configuration

The serial BIOS on the testbed nodes is upgraded to version 1.28. Additionally the console speed is set to 115200 bps and the boot order is set to network boot followed by booting from the local disk. This boot order is important because it allows a way to recover nodes that experience sporadic corruption to the system image. It forces the boxes to first attempt a PXE boot, at which point the box can be instructed to reinstall the system image, regardless of the state of it's current image.

The BIOS command sequence for setting these parameters is:

```
set ConSpeed=115200
set BootDevice=F0 01 02 00
```

It is worth noting that new boxes ship from the factory with a console speed of 19200.

In addition to the configuration there is one anomaly in the BIOS that is worth mentioning. After a warm reboot the PXE network boot loader *always* fails to contact the DHCP server. This means network boot never works after a soft reboot. If network boot is desired a hard reboot, using the PoE control scripts described in A.6.5, must be used.

A.4.8 Custom Software Configurations

The testbed nodes contain a number of custom configuration and configuration scripts that keep them operating. This section details a number of the most important ones.

The initial configuration of a node's radios is controlled by the `ath[01]_config.sh` script found in the `/ro/root/scripts` directory. These scripts are run once during the startup process of the node, and should be run on completion of any experiments. This guarantees the radios are always in a known good state.

The testbed nodes include a hardware watchdog. Briefly, a hardware watchdog is additional circuitry, either on the logic board or in the CPU, that reboots the node at a fixed interval. In order to prevent rebooting, the CPU must periodically clear a register. When OS crashes it gets detected after a small period of time and the box rebooted. Support for the watchdog requires a kernel extension and corresponding user level application. The application `/usr/sbin/watchdog` runs periodically and requests that the kernel reset the watchdog state. The watchdog driver is compiled in to the custom version of the kernel used in the testbed.

NTP [Aut] is used to keep the system clocks on the testbed nodes roughly synchronized. For some purposes, like monitoring the status of the nodes, this level of synchronization is sufficient. However some projects require more accurate synchronization and can use the NTP system clock as a starting point to reduce the complexity of attaining the necessary accuracy. In addition there is a cronjob that runs `/ro/root/scripts/save_ntpdriфт.sh` every 30 minutes. The script remounts the flash read-write, saves the current NTP drift file, and then remounts it read-only. This prevents the measured drift value from being completely lost when the node reboots.

In addition to the standard set of metrics reported by ganglia there is a script that runs on the testbed nodes to report more specific information of interest. The name of the script is `report_clients`, and it is located in `/ro/root/scripts/report_clients.sh`. `report_clients` is started from an `init.d` script at the appropriate run levels. New metric values are reported every 10 seconds. Although its name might suggest otherwise this

scripts reports a large number of additional metrics as follows:

- `associated_clients_ath0` – The number of clients currently associated with the `ath0` interface. This value is reported by the driver and read via the `proc` filesystem.
- `associated_clients_ath1` – The number of clients currently associated with the `ath1` interface.
- `status_code` – A numerical code for the nodes current action. See the description of custom status reporting in section A.4.9 for more details.
- `status_string` – A textual description of the status code. See the description of custom status reporting in section A.4.9 for more details.
- `ntp_offset` – The difference between the node’s wall clock and NTP time as reported by the server `bigben.ucsd.edu`. This data enables monitoring of the difference, and flagging large values. The quality of the NTP synchronization, down to the sub-millisecond level, is important for some of the experiments run on the testbed.
- `ntp_drift` – The value contained in the NTP drift file. Storing this values allows it to be restored when a node is reimaged.

In addition to directly reporting these metrics, the `report_clients` script calls the `/root/scripts/dev_detailed_stats.pl` perl script to report summary statistics on the wireless cards. These additional metrics include the transmission and reception rate, in units of bytes and packets per second, since the last time values were reported. The numbers are reported for all interfaces.

In addition to the script running as daemons, there are three periodic tasks managed by `cron`. The first task calls `/ro/root/scripts/apscan.sh` 6 times an hour. `Ap-scan` instructs the wireless cards to perform a full spectrum scan, and emails the scan results to a single address for consolidation. However, to prevent interrupting long running experiments, the script first checks for the presence of `/tmp/ath0scan.lock` and

/tmp/ath1scan.lock. If these files exist the scanning action on the associated interface is skipped.

The second periodic job is pruning old trace files from the built-in hard drives, and is accomplished via the `/root/scripts/cleanstats_cron.sh` script. This is legacy code because the hard drives have been removed from the nodes for some time now. However it does still run and hence needed documentation.

The last cron job is `/root/scripts/save_ntpdriфт.sh`. As its name implied, this script persistently saves the NTP drift file to the flash memory so its contents are not lost across reboots.

A.4.9 Custom Status Reporting

One important custom metric reported by the testbed nodes is status. Status reporting enables monitoring of the nodes progress during image reinstallation, and provides an easy way to determine if all the nodes are running the correct process for a given experiment. Status is reported in two parts, a numeric status code and a corresponding status string. The code is authoritative and intended for use in monitoring logic. The string is intended for visualization purposes only, eliminating the visualization tool from the burden of knowing all the various status codes *a priori*. The status code is written as an ASCII number into the file `/tmp/status.code`, and the status string into `/tmp/status.string`.

The installation script makes heavy use of the status string, which in turn allows a visualization tool to display the exact reinstallation progress for all nodes. The strings reported by the installation process progress from formatting, through installing base image, pruning image, installing custom files, installing custom devices, installing bootloader, installing machine specific files, computing dependencies, and rebooting. See section A.4.2 for a detailed description of the installation process.

A.4.10 Customized Atheros Driver Information

Each different project that utilizes the testbed typically requires a different customized Madwifi [Mad] driver. The drivers used for the work presented in this thesis can be found in `/projects/wireless-aps/soekris/pebble/pebble-v36/kernel-2.4.31-ucsd-soekris/madwifi-cvs`. The details of driver customizations are located in their respective chapters, however it is worth noting that the customizations typically require modifying the supporting user level applications with appropriate logic to understand the newly added metrics.

A.4.11 Capture Portal

One of the motivations behind deploying such a large wireless testbed is attracting a diverse user population. The users in turn generate a realistic workload, which creates more confidence in experimental results. However there was substantial concern within our department's computer support staff that we would snare unwitting users. People who did not know they were using the testbed, or people who simply were capable of forcing their machine to associate with the production network in the building. I developed a capture portal solution to address these concerns.

A capture portal is a router that intercepts all web traffic and redirects to a "login" page. They are customarily found on networks that charge users for access, like coffee shop wireless networks. Because of their prevalent use a number of open source solutions exist, however none were directly applicable to my needs.

There are two main differences between the testbed and more common capture portal installations. First, the testbed nodes do not actually route traffic, they merely bridge traffic to the campus network which enables campus networking staff to enforce access restrictions. All the current solutions are designed with the assumption they operate on the network's router. Second, once a user was captured and agreed to use the network, I wanted to ensure they were not captured again, even if they roamed to a different AP. The existing capture portals are focused on topologies where all traffic, regardless of which AP the client is associated with, pass through the same router. To

solve these problems I heavily modified NoCatSplash, version 0.93pre2, as outlined below.

Instead of using the standard linux iptables (routing firewall ruleset) I wrote a ruleset that works with ebtables, linux's bridge firewall ruleset. The general layout of the rules places MAC addresses into three groups, *authorized*, *unauthorized*, and *limbo*. By default all nodes are in limbo. A node moves from limbo to authorized or unauthorized by accepting or declining to use the network when presented with a splash page. However the ability to redirect traffic within the linux kernel still relies on the routing ruleset. So all web requests from limbo clients are forced into the routing system while all other frames are bridged normally. This is referred to as *brouting*.

To achieve the goal of mobility the splash page contents needed to be hosted on a central server. This created a whole separate set of complexities. The routing control for all client traffic is outside the control of the AP. In fact, the campus uses a capture portal themselves (at the router level) to control network access. If a client is not authenticated to the campus it will not be able to display the splash page for the testbed network, which creates a chicken-and-egg problem.

Solving this problem required additional use of the brouter functionality. In addition to capturing all web traffic for clients in limbo, the AP maintains a list of sites the client is allowed to access via network address translation (NAT). When a web page from one of those sites is requested, the frame gets promoted to the routing layer and NATed. In addition, a specialty ebtables action was written to add the MAC address to a NATed hosts list. This is required to correctly de-NAT response packets from the server.

To solve the second problem, roving client state, NoCatSplash was modified to store and query client state from a central database. On the surface this change is straightforward, however there is one complication, the node has no of of knowing which clients will associate with it before the association takes place. Without knowing which client are going to show up it is impossible to keep the first connection of a new client out of limbo, which results in the connection being captured. To mitigate this problem I developed a second ebtable action. This action reported the MAC address of

all clients in limbo via a special device file. The first frame a client sends upon associating with an AP in our network is a DHCP request. The MAC address of the client is reported when this DHCP request is seen. This, in turn, allows the NoCatSplash program to perform a speculative lookup on the client and transition them into authorized as appropriate. The lookup happens in parallel with the client's DHCP exchange, and, since it takes a very small amount of time, an authorized client's first connection doesn't get mistakenly stuck in limbo.

The source for the modified NoCatSplash is in `/projects/wireless-aps/soekris/pebble/pebble-v36/apps/NoCatSplash-0.93pre2` and the source for the custom ehtable filters is in the kernel tree, `/projects/wireless-aps/soekris/pebble/pebble-v36/kernel-2.4.31-ucsd-soekris/linux-2.4.31-ucsd-soekris`.

A.4.12 Initial configuring a new node

A number of settings need to be changed in order to add a new node to the testbed. In order, the steps necessary to bring a new node online are as follows:

1. Set up a serial connection to the node with an initial console speed of 19200
2. power on the node
3. enter the BIOS by pressing ctrl-p when prompted
4. Download and install version 1.28 of the firmware via the following command sequence:
 - (a) download -
 - (b) Initiate a xmodem send of firmware image from the serial client
 - (c) flashupdate
5. set ConSpeed=115200
6. set BootDrive=F0 80 81 00
7. reboot

8. change console speed to 115200 on the serial client
9. enter the BIOS by pressing ctrl-p when prompted
10. let PXE boot start and record the MAC address of the ethernet interface
11. power off the node
12. add the MAC address to wireless.ucsd.sys.net:/projects/wireless-aps/ap-list/aplist.txt
13. run “sudo make” in /projects/wireless-aps/ap-list on wireless.ucsd.sys.net
14. power on testbed node
15. boot the “install” image when presented with the PXE prompt

After the software installation is complete the node is ready to be added to the testbed.

A.5 Testbed Interconnect

In order for the testbed to be by the campus administrators approved it was necessary to ensure all clients are subject to campus network access restrictions. The easiest way to achieve this is to give the campus network administration control over the client’s frames. This was accomplished by deploying two VLAN to each AP. The first VLAN is a management network. The AP has an IP address on the management network, and all command and control is performed thusly. To facilitate network booting this network needs to be accessible before the kernel is loaded, therefore the frames on the management VLAN are delivered untagged to the APs.

The second VLAN available to the APs is connected to network operations. This VLAN is delivered over the same ethernet link as the management VLAN, however the frames are tagged according to the 802.1q standard. All traffic from clients associated to the testbed APs is bridged directly to this VLAN, with the exception of traffic destined for the testbed’s capture portal as described in section A.4.11. This gives

the campus network administrators complete admission control over the wireless clients connected to the testbed.

A.5.1 Switch Configuration

The switches in the testbed require a special configuration, and this section details what is necessary to configure them. Each node is on two different 802.1q VLANs, a management VLAN and a wireless client VLAN. The management VLAN is delivered as the default (untagged) VLAN. This allows the nodes to obtain IP addresses via DHCP and participate in network booting, among other features. The wireless client VLAN is delivered using tagged frames.

The uplink is configured for three tagged VLANs, the management VLAN, wireless client VLAN, and the systems VLAN. The two 1-gigabit ports are aggregated using link aggregation control protocol (LACP) which both increases the available uplink bandwidth to 2 gigabits and provides fault tolerance in the face of faultily wiring.

The HP switches used in the infrastructure support a variety of configuration techniques including serial, telnet, http, and SNMP. The next series of instructions details how to set up a new switch, taking into account the order in which the options must be set.

1. Connect to the serial port on the switch and power it on. Do not connect it to any uplink links. The serial port auto detects the baud rate, just hit enter a few times to get the login prompt. Initially there is no password on the switch.
2. Enter the menu configuration tool (“menu” option from command line). Within this tool set the following options in order. The numbers immediately preceding the option name indicate which menu items need to be selected to reach the appropriate screen. For example, 2->1->NTP is interpreted as selecting item 2 in the first menu, item 1 in 2’s submenu, and then the NTP option within the sub-submenu.
 - (a) 2->1->NTP settings (method SNTP, mode unicast, server address

132.239.1.6, time zone -490, daylight time rule : continental-US-and-Canada)

(b) 2->8->Vlan Names

=> add 115 "80211-client"

=> add 439 "wireless-mgt"

=> add 441 "systems"

(c) 2->8->Vlan Support (Primary Vlan => "wireless-mgt")

(d) 2->Port/Trunk Settings

=> make group "Trk1" and type "LACP" on ports 25 and 26

(e) 2->8->Vlan Port Assignment

=> Make ports 1-24 (default_vlan: no, 80211-client: Tagged, wireless-mgt: untagged, systems: no)

=> make Trk1 (default_vlan: no, 80211-client: tagged, wireless-mgt:untagged, systems: tagged)

(f) 2->5->IP Settings

=> Disabled for all VLANs expect:

=> Manual for wireless-mgt with appropriate IP and netmask

=> gateway: 172.22.13.1 (leave IP routing disabled)

(g) 2->SNMP Community Names

=> Delete public

=> add wireless SNMP community (look at another switch for name; omitted for security reasons)

3. Plug the switch into the uplink connection and then go back to the command line configuration interface (i.e., leave the menu interface). Enter the command:

> configure web-management plaintext

4. Open up a web browser and browse to the switch's IP address. In order, use the following screens to set the options:

- (a) Security->SSL tab
 - =>create new SSL cert
 - =>enable SSL
- 5. Verify SSL is working by reconnecting to the switch with https instead of http
- 6. Switch back to the command line interface and issue the following directive:
 - > configure no web-management plaintext
- 7. Switch back to the browser's SSL configuration session and perform the following:
 - (a) Security->Device Passwords
 - set "read-write access" username and password to the correct values for the installation
- 8. Switch back to the menu configuration interface. This is done by running the menu command from the command line interface. Change the following:
 - (a) 2->IP Authorized Managers
 - => add 172.22.13.254/24
- 9. Reboot Switch

Switch configuration is now complete.

A.6 Wireless Control Machine

There is a control machine for the testbed in addition to the nodes. The control machine is responsible for running all the necessary services, maintaining the master node image, and for building packages that get deployed to the testbed nodes. The control machine is a critical part of the testbed infrastructure, and failure of this machine prevents the testbed from operating. This section discusses the important aspects of the control machine.

A.6.1 Connectivity

The control machine is allocated two different IP addresses on two different virtual local area networks (VLANs). The first address is a private address on the VLAN occupied only by the testbed nodes. This connection enables the control machine to use direct broadcast protocols, such as DHCP, to coordinate the testbed nodes. The second connection is to the public network at UCSD. This permits access from outside hosts to services running on the control machine. The Linux kernel supports 802.1q VLAN tagging, so it is possible to provide access to both VLANs with one physical network cable. Example syntax for enabling VLANs in Linux is:

1. `vconfig add eth0 439`
2. `vconfig set_flag eth0.439 1 1`
3. `ifconfig eth0.439 up`

A.6.2 Compilation

The library and header environment on the control machine is set up to closely mimic the environment that runs on the testbed nodes. This allows programs to be linked with dynamic libraries. The other alternative, static linking, results in executable sizes on the order of 5 megabytes each. Since each node only has 64 megabytes of storage 5 megabyte executables would dramatically limit the usefulness of the nodes.

A.6.3 Services

The control machine provides a number of critical services for the testbed nodes. One of the most important services is DHCP. Each machine in the testbed has a separate DHCP entry, keyed by MAC address. This is important for two reasons. First, it reduces the complexity of configuring the individual nodes. Secondly DHCP is required to perform network booting through PXE. Network booting it used to install and update the OS images on the testbed nodes.

Trivial file transfer protocol (TFTP) support is also necessary to enable network booting. Therefore the control machine also provides TFTP service to the testbed subnet. Only the minimum files necessary for network booting are available via TFTP. These files include the pxelinux bootloader and related configuration files.

The wireless control machine also provides DNS service. This is necessary to handle the sometimes high amount of churn as new nodes are added and older ones relocated. The campus DNS mechanisms are manual and therefore can't be automated. Both forward and reverse service is provided. The wireless control machine is the only DNS server the testbed nodes know about.

The DNS and DHCP configuration files need to be updated every time there is a change in the testbed composition. This process has been automated to reduce human error and to speed up turnaround time. The makefile in `/projects/wireless-aps/ap-list` recreates all the necessary configuration files after consulting the node installation database. The database is discussed in section A.6.4.

Ganglia [The] is used to monitor the health of the testbed nodes. Its design requires that a single server aggregate the statistics from the individual nodes. The control machine is the obvious place to provide this service. As a result, the detailed node status for the entire testbed is available in XML format from a TCP port on the control machine.

The remaining services running on the control machine are primarily in support of ongoing research, as opposed to support of the testbed. Some of these services include a web server and an outbound mail server. The exact composition of such services varies over time as the specific experimental needs for a project evolve.

A.6.4 Node Installation Database

A number of the tools and scripts require accurate information about a particular node's installation characteristics, including its MAC address and which switch port it is connected to. In order to provide this information it is necessary to maintain a database and update it when changes to the testbed have been made. The database is

located at wireless.ucsd.sys.net/projects/wireless-aps/ap-list/aplist.txt, and is formatted as a comma separated text file. The first line of the file provides the name for each subsequent field. Some of the available information includes MAC address of all wired and wireless interfaces, IP address, node name, and the switch name and port. To enable tools that run on a machine other than the control machine the current version of the database is also available at http://wireless.ucsd.sys.net/aplist/sysnet_ap_database.txt.

A.6.5 Other useful management tools

The control machine includes a set of scripts that help manage the power-over-ethernet capabilities of the switches. These scripts are located in `/projects/wireless-aps/switch-management/bin` and are called `power_on`, `power_off`, and `power_cycle`. They accept the name of the AP as the sole parameter and take the specified action. They work by sending an appropriate SNMP request to the correct switch. The scripts look up the correct switch and port for a given AP using the node database.

In addition to the PoE scripts there is a script that controls the default startup sequence of a node. This script is named `default_boot`, and is also located in `/projects/wireless-aps/switch-management/bin`. The first argument is the name of a pxelinux configuration file, which must already be present in the `/tftp/pxelinux.cfg` directory. The second argument is the name of an AP. The script arranges for the AP to load the specified configuration file during the pxelinux phase of the boot chain. This is typically used to change the default boot behavior from `install` to `localboot` (or vice versa), however the configuration file is very expressive resulting in a wide range of possible options.

The `/projects/wireless-aps/ap-management/list_aps.pl` script returns the names of all active APs. It does so by querying the ganglia data to determine the last time a node reported statistics, and then displays all the nodes that have reported recently. This is useful when developing automated experiment scripts that need to know which nodes are available while, at the same time, allowing the set of nodes used in an experiment to evolve over time.

Bibliography

- [AA02] A.Mishra and W.A. Arbaugh. An Initial Security Analysis of the IEEE 802.1X Standard. Technical Report CS-TR-4328, University of Maryland, College Park, February 2002.
- [AC] Inc. Apple Computer. Bonjour software package. <http://www.apple.com/macosx/features/bonjour/>.
- [ALM⁺92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.
- [ANJ01] W.A. Arbaugh, N.Shankar, and J.Wang. Your 802.11 Network has no Clothes. In *First IEEE International Conference on Wireless LANs and Home Networks*, pages 131–144, December 2001.
- [Aut] Various Authors. Network Time Protocol. <http://www.ntp.org/>.
- [BC00] H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. In *USENIX - LISA*, New Orleans, LA, December 2000.
- [BDSZ94] Vaduvur Bharghavan, Alan J. Demers, Scott Shenker, and Lixia Zhang. MACAW: A media access protocol for wireless LAN's. In *SIGCOMM*, pages 212–225, 1994.
- [BGW01] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In *Seventh Annual International Conference on Mobile Computing And Networking*, July 2001.
- [BL02] Robert Baird and Mike Lynn. Advanced 802.11 Attack. BlackHat Presentation, July 2002. <http://www.blackhat.com/presentations/bh-usa-02/baird-lynn/bh-us-02-lynn-802.11attack.ppt>.
- [Blo] Google WiFi Blog. <http://googleblog.blogspot.com/2005/11/wi-fi-in-mountain-view.html>.
- [BRBB05] Vladimir Brik, Eric Rozner, Suman Banerjee, and Paramvir Bahl. Dsap: A protocol for coordinated spectrum access. In *IEEE Dynamic Spectrum Access Networks (DySPAN)*, Baltimore, MD, November 2005.

- [CBB⁺06] Yu-Chung Cheng, John Bellardo, Peter Benko, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. In *SIGCOMM*, 2006.
- [CFSD] J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC 1067 – A Simple Network Management Protocol. <http://www.rfc-archive.org/getrfc.php?rfc=1067>.
- [Com] Wireless Philadelphia Executive Committee. <http://www.phila.gov/wireless/>.
- [FMS01] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of rc4. *Lecture Notes in Computer Science*, 2259, 2001.
- [Gaz] Linux Gazette. Improving Hard Disk Performance with hdparam. <http://linuxgazette.net/issue79/punk.html>.
- [GKF02] Vikram Gupta, Srikanth Krishnamurthy, and Michalis Faloutsos. Denial of Service Attacks at the MAC Layer in Wireless Ad Hoc Networks. In *MILCOM - Network Security*, Anaheim, CA, October 2002.
- [Gro] The Dell’Oro Group. <http://www.delloro.com/>.
- [Hew] Hewlett-Packard Development Company, L.P. Switch 2626-PWR (J8164A). <http://www.hp.com/rnd/products/switches/switch2600series/overview.htm>.
- [Hom] Homer TLC, Inc. OOK Anti-Theft Security Hanger Model 50044PN, Internet Number 908870, Catalog Number 100070729. <http://www.homedepot.com/>.
- [HS65] J. Hartmanis and R. E. Stearns. On the Computational Complexity of Algorithms. *Trans. Amer. Math. Soc.*, 117(5):285–306, May 1965.
- [KMS94] David R. Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. In *IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1994.
- [KNG⁺04] D. Kotz, C. Newport, R. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions, 2004.
- [KV02] P. Kyasanur and N. Vaidya. Detection and handling of mac layer misbehavior in wireless networks. Technical Report CSL, University of Illinois at Urbana-Champaign, August 2002.
- [KWBF03] Bhaskar Krishnamachari, Stephen Wicker, Ramon Bejar, and Cesar Fernandez. On the Complexity of Distributed Self-Configuration in Wireless Networks. *Journal of Telecommunication Systems*, 22(1–4):33–59, January 2003.

- [Lam96] B. W. Lampson. How to build a highly available system using consensus. In Babaoglu and Marzullo, editors, *10th International Workshop on Distributed Algorithms (WDAG 96)*, volume 1151, pages 1–17. Springer-Verlag, Berlin Germany, 1996.
- [LKC02] Youngseok Lee, Kyoungae Kim, and Yanghee Choi. Optimization of AP Placement and Channel Assignment in Wireless LANs. In *IEEE Conference on Local Computer Networks*, 2002.
- [Lou01] Daniel Lowry Lough. *A Taxonomy of Computer Attacks with Applications to Wireless Networks*. PhD thesis, Virginia Polytechnic Institute and State University, April 2001. <http://scholar.lib.vt.edu/theses/available/etd-04252001-234145>.
- [Mad] Madwifi Atheros Linux Driver. <http://madwifi.org/>.
- [MBA05] Arunesh Mishra, Suman Banerjee, and William Arbaugh. Weighted Coloring based Channel Assignment in WLANs. *Mobile Computing and Communications Review*, July 2005.
- [MBB⁺06] Arunesh Mishra, Vladimir Brik, Suman Banerjee, Aravind Srinivasan, and William Arbaugh. A client-driven approach for channel management in wireless lans. In *IEEE Conference on Computer Communications (INFOCOM)*, Barcelona, 2006.
- [Mid] Midcom, Inc. Discrete Single Port 10/100 Base-T PDSO-G16. http://www.midcom-inc.com/Products/Lan/D_SNGL_100_PDSO_G16.asp.
- [MMS03] Filipe F. Mazzini, Geraldo R. Mateus, and James MacGregor Smith. Lagrangean Based Methods for Solving Large-Scale Cellular Network Design Problems. *Journal of Wireless Networks*, 9(6):659–672, November 2003.
- [Moh] B. Mohar. Circular colorings of edge-weighted graphs.
- [Net] Aruba Wireless Networks(TM). Aruba wireless networks delivers painless wi-fi to sharp healthcare, http://www.prnewswire.com/cgi-bin/stories.pl?acct=cmptw_tel.story&story=/www/story/07-19-2004/0002212557&edate=mon%20jul%2019%202004,%2008:07%20am.
- [NYCa] NYC Wireless. Pebble README. <http://nycwireless.net/pebble/pebble.README>.
- [NYCb] NYCwireless. <http://www.nycwireless.net/>.
- [NYCc] NYCwireless. Pebble Linux distribution. <http://www.nycwireless.net/pebble/>.

- [RcC05] Ashish Raniwala and Tzi cker Chiueh. Architecture and Algorithms for an IEEE 802.11-Based Multi-Channel Wireless Mesh Network. In *IEEE Infocom*, 2005.
- [RGcC04] Ashish Raniwala, Kartik Gopalan, and Tzi cker Chiueh. Centralized algorithms for multi-channel wireless mesh networks. In *ACM SIGMOBILE Mobile Computing and Communications Review (MC2R)*, 2004.
- [Sim] NS Network Simulator. <http://www.isi.edu/nsnam/ns/>, version 2.1b9a.
- [SIR02] Adam Stubblefield, John Ioannidis, and Aviel Rubin. Using the fluhrer, mantin, and shamir attack to break wep. In *Proceedings of the 2002 Network and Distributed Systems Symposium*, San Diego, CA, February 2002.
- [SLM92] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.
- [Soc99] IEEE Computer Society. Wireless lan medium access control (mac) and physical layer(phy) specification. iee standard 802.11, 1999 edition., 1999.
- [Soc01] IEEE Computer Society. Port based network access control. iee std 802.1x, 2001 edition., June 2001.
- [Soea] Soekris Engineering. <http://www.soekris.com/>.
- [Soeb] Soekris Engineering. net4801. <http://www.soekris.com/net4801.htm>.
- [Soec] Soekris Engineering. net4826. <http://www.soekris.com/net4826.htm>.
- [Sto00] R. Stone. Centertrack: An ip overlay network for tracking. In *USENIX Security Symposium*, Denver, CO, July 2000.
- [SWKA00] Stefan Savage, David Wetherall, Anna R. Karlin, and Tom Anderson. Practical network support for IP traceback. In *SIGCOMM*, pages 295–306, 2000.
- [Teaa] Net-SNMP Team. Net-SNMP software package. <http://net-snmp.sourceforge.net/>.
- [Teab] OpenSLP Team. OpenSLP software package. <http://www.openslp.org/>.
- [The] The Ganglia Team. <http://ganglia.sourceforge.net/>.
- [Tou] Jean Tourrilhes. Wireless Tools for Linux. <http://hplabs.hp.com/personal/Jean.Tourrilhes/Linux/Tools.html>.

- [Web] Sharp HealthCare Website. <http://www.sharphealthcare.com/>.
- [Wir] Seattle Wireless. NoCatSplash software package. <http://www.seattlewireless.net/NoCatSplash>.