UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Automating cross-layer diagnosis of enterprise 802.11 wireless networks

Permalink

<https://escholarship.org/uc/item/3rv3w0f3>

Author

Cheng, Yu-Chung

Publication Date

2007

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

**Automating Cross-Layer Diagnosis of Enterprise 802.11 Wireless
Networks**

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

by

Yu-Chung Cheng

Committee in charge:

Professor Geoffrey M. Voelker, Co-chair
Professor Stefan Savage, Co-chair
Professor Rene L. Cruz
Professor Tara Javidi
Professor Alex C. Snoeren

2007

Copyright
Yu-Chung Cheng, 2007
All rights reserved.

The dissertation of Yu-Chung Cheng is approved, and it is
acceptable in quality and form for publication on microfilm:

Co-chair

Co-chair

University of California, San Diego

2007

TABLE OF CONTENTS

	Signature Page	iii
	Table of Contents	iv
	List of Figures	vi
	List of Tables	viii
	Acknowledgments	ix
	Vita	xi
	Abstract	xii
Chapter 1	Introduction	1
	1.1. Complexities in the enterprise 802.11 environment	2
	1.2. The challenge of diagnosing enterprise 802.11 problems	3
	1.3. Contributions	4
	1.3.1. Large-scale distributed network monitoring and synchro- nization.	5
	1.3.2. Cross-layer protocol states modelings.	6
	1.3.3. Real-time automated root-cause diagnosis	6
	1.4. Overview of the dissertation	7
Chapter 2	Background	8
	2.1. The 802.11 wireless network protocols	8
	2.1.1. Network management and configurations	8
	2.1.2. MAC channel access	9
	2.1.3. PHY implementations	10
	2.2. Wireless network measurements and characterizations	11
	2.3. Critical path analysis of networking problems	12
	2.4. Wireless network diagnosis systems	13
	2.5. Summary	15
Chapter 3	Shaman System Architecture	16
	3.1. Monitor infrastructure	16
	3.2. Synchronization and preprocessing	17
	3.2.1. Protocol states inference	18
	3.3. Critical path diagnosis tools	19
	3.4. Acknowledgement	20

Chapter 4	Network Monitoring and Trace Synchronization	21
	4.1. Monitor infrastructure	21
	4.1.1. Environment	22
	4.1.2. Hardware	23
	4.1.3. Software	24
	4.1.4. Coverage	25
	4.1.5. Monitoring performance and overhead	29
	4.2. Trace unification and synchronization	29
	4.2.1. Bootstrap synchronization	31
	4.2.2. Frame unification	33
	4.3. Link-layer conversations reconstruction	40
	4.3.1. Station module	40
	4.3.2. Link-layer inference	41
	4.3.3. Inference from TCP layer	43
	4.3.4. Cross reference of wired and wireless frames	44
	4.4. Acknowledgement	46
Chapter 5	Protocol Models and Analysis	47
	5.1. Trace summary	48
	5.2. The troubled life of a packet	51
	5.2.1. Physical layer	51
	5.2.2. Link layer	53
	5.2.3. Infrastructure support	55
	5.2.4. Transport layer	55
	5.3. Modeling 802.11 frame loss	55
	5.3.1. Hidden-terminal interference	56
	5.4. Modeling 802.11 frame delay	59
	5.4.1. Critical path timestamps	60
	5.4.2. Critical path delays	64
	5.4.3. Validating the model	64
	5.4.4. Applying the model	69
	5.5. 802.11b/g compatibility overhead	70
	5.6. TCP performance analyzer	73
	5.6.1. Interactive TCP flow analysis	73
	5.6.2. Bulk TCP flow analysis	75
	5.7. Shaman diagnostic tools	78
	5.7.1. User diagnostic tool	78
	5.7.2. Network alert tool	80
	5.8. Acknowledgement	81
Chapter 6	Conclusion	83
	Bibliography	85

LIST OF FIGURES

Figure 3.1	The Shaman system architecture.	17
Figure 4.1	UCSD CSE building floor plan. This building comprises roughly 150,000 square feet over four floors and a basement. Circles indicate wireless sensor pods, and triangles indicate production access points.	22
Figure 4.2	Coverage of frames transmitted by clients and APs.	26
Figure 4.3	Coverage of frames transmitted by APs and clients for various configurations of sensor pods in the building.	28
Figure 4.4	Visualization of synchronized trace. Time appears on the x-axis in μs and individual radios on the y-axis. At 400 μs , six radios receives a data frame sent by a client and five of them, except p4450, receives the corresponding ack	34
Figure 4.5	Frame unification among five radios. Shaman (a) uses a search window to identify candidate instances; (b) unifies identical instances into a frame; (c) adjusts the corresponding radio clocks; (d) unifies the white instances into the next frame. . .	35
Figure 4.6	CDF of group dispersion across all frames.	39
Figure 4.7	Finite state machines used to assemble individual frames into transmission attempts (left) and to compose transmission attempts into complete frame exchanges (right).	42
Figure 5.1	Time series of network activity throughout the day in one-minute intervals.	49
Figure 5.2	Physical error frame pattern during microwave oven use. The y-axis depicts (time % 16) ms, showing the offset of physical error packets within the 16 ms microwave period. The x-axis shows two minutes of microwave use.	52
Figure 5.3	Interference loss rate X across (s, r) pairs.	58
Figure 5.4	Representation of wired network, queuing and transmission behavior in the AP. The arrows indicate where network we measure/infer timestamps as frames traverse the network. . .	61
Figure 5.5	Simplified 802.11 DCF operation for unicast.	65
Figure 5.6	Access delay (d_{t0}) distribution of one hour of head-of-line blocked frame exchanges from an Avaya AP-8 AP to a client doing a bulk TCP download.	67
Figure 5.7	Access delay (d_{t0}) distribution of head-of-line blocked frame exchanges from a Cisco Aironet 350 AP to a client doing a bulk scp download.	68
Figure 5.8	Critical path delays, goodput, and losses over time for frame exchanges from an AP to a client doing a bulk TCP download. . .	69

Figure 5.9	Overprotective APs and active 802.11g clients during the busy period of the trace.	72
Figure 5.10	Response time and TCP behavior of a user who reported substantial delay with an interactive SSH session.	74
Figure 5.11	Root wireless causes that limit TCP flow throughput.	77
Figure 5.12	Shaman problem report form.	79
Figure 5.13	Shaman basic diagnosis.	80

LIST OF TABLES

Table 5.1	Summary of trace characteristics.	48
Table 5.2	Summary of timestamps and delays determined by the media access model.	62

ACKNOWLEDGMENTS

I have a fantastic and fruitful Ph.D. process because I have the best Ph.D. advisors in the world: Stefan Savage and Geoff Voelker. I am very lucky to have met them on the first day in UCSD CSE. They are knowledgeable, rigorous, vigorous, and most importantly, always coherent and consistent on the directions of my Ph.D. research. Both of them teach me how to identify important problems and conduct scientific research, and also how to fun doing it. I admire Stefan's wit and ideas about the technologies and the research. In particular, he listens to my random ideas and gives enlightening feedbacks. I remember he was late for a Friday family dinner party because he stayed in the office to discuss my ideas. On the other hand, I have never seen another teacher who cares more about students other than Geoff. I still remember once Geoff came to the office on weekend to help me debug my program. His advises are always very encouraging and I feel truly secure and confident that I will succeed because Geoff is always there to help.

In addition, I can not expect a more exciting research environment in the systems and networking group at UC San Diego. Among all my colleagues, I would like to first thank Prof. Alex Snoeren for participating all my research meetings and giving insightful advices. John Bellardo helped in the early stages of my dissertation work. Without John's help building the monitoring platform and fixing bugs in Jigsaw, I would not have published my first SIGCOMM paper. Next I would like to thank Mikhail Afanasyev and Patrick Verkaik for their energetic support in my second SIGCOMM paper. Besides, Barath Raghavan, Ishwar Ramani, and Sumeet Singh are my favorite lab-buddies to discuss research ideas. Alper Mizrak and I spent many Friday nights together in the lab when we were both junior Ph.D. students. I appreciate that he invited me to participate in his research projects.

I could not have finished my Ph.D. without Frances. Her encouragement kept me making progress when I struggled. I am grateful for her patience for letting me spend many weekends in the lab. Leo introduced golf to me and I keep a balanced, healthy life thanks to our weekly golf rounds. Lastly, I am extremely grateful for my family back in

Taiwan for providing the full support throughout the graduate study.

Chapter 3, 4 and 5, in part, are a reprint of the material as it appears in the SIGCOMM Conference, 2006, Cheng, Yu-Chung; Bellardo, John; Benko, Peter; Snoeren, Alex C.; Voelker, Geoffrey M.; Savage, Stefan. The dissertation author was the primary investigator and author of this paper.

Chapter 3, 4 and 5, in part, are a reprint of the material as it appears in the SIGCOMM Conference, 2007, Cheng, Yu-Chung; Afanasyev, Mikhail; Verkaik, Patrick; Benko, Peter; Chiang, Jennifer; Snoeren, Alex C.; Voelker, Geoffrey M.; Savage, Stefan. The dissertation author was the primary investigator and author of this paper.

VITA

1998	Bachelor of Science in Engineering National Taiwan University Taipei, Taiwan
2004	Master of Science in Computer Science University of California, San Diego San Diego, CA, USA
2007	Doctor of Philosophy in Computer Science University of California, San Diego San Diego, CA, USA

PUBLICATIONS

- "Automating Cross-Layer Diagnosis of Enterprise Wireless Networks".
Y. Cheng, M. Afanasyev, P. Verkaik, P. Benko, J. Chiang, A. Snoeren, G. Voelker, and S. Savage. *SIGCOMM*, 2007
- "Jigsaw: Solving the puzzle of enterprise 802.11 analysis".
Y. Cheng, J. Bellardo, P. Benko, A. Snoeren, G. Voelker, and S. Savage. *SIGCOMM*, 2006
- "Detecting and Isolating Malicious Routers".
A. T. Mizrak, Y. Cheng, K. Marzullo, and S. Savage. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, Vol. 3, No. 3, 2006
- "Fatih: Detecting and Isolating Malicious Routers".
A. T. Mizrak, Y. Cheng, K. Marzullo, and S. Savage. *DSN*, 2005
- "Accuracy Characterization of Metropolitan-scale Wi-Fi Localization".
Y. Cheng, Y. Chawathe, A. LaMarca, and J. Krumm. *MobiSys*, 2005
- "Monkey See, Monkey Do: A Tool for TCP Tracing and Replaying".
Y. Cheng, U. Hölzle, N. Cardwell, S. Savage, and G. Voelker. *USENIX*, 2004
- "Total Recall: System Support for Automated Availability Management".
R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker. *NSDI*, 2004
- "Structuring Super-peers: Leveraging Heterogeneity to Provide Constant Time Lookup".
A. T. Mizrak, Y. Cheng, V. Kumar, and S. Savage. *IEEE Workshop on Internet Applications (WIAPP)*, 2003

FIELDS OF STUDY

Computer Systems and Networking

ABSTRACT OF THE DISSERTATION

Automating Cross-Layer Diagnosis of Enterprise 802.11 Wireless Networks

by

Yu-Chung Cheng

Doctor of Philosophy in Computer Science

University of California, San Diego, 2007

Professor Geoffrey M. Voelker, Co-Chair

Professor Stefan Savage, Co-Chair

The combination of unlicensed spectrum, cheap wireless interfaces and the inherent convenience of untethered computing have made 802.11-based networks ubiquitous in the enterprise. Modern universities, corporate campuses and government offices routinely deploy scores of access points to blanket their sites with wireless Internet access. However, while the fine-grained behavior of the 802.11 protocol itself has been well studied, our understanding of how large 802.11 networks behave in their full empirical complexity is surprisingly limited.

Enterprise networks are of sufficient complexity that even *simple* faults can be difficult to diagnose — let alone transient outages or service degradations. Nowhere is this problem more apparent than in the 802.11-based wireless access networks now ubiquitous in the enterprise. In addition to the myriad complexities of the wired network, wireless networks face the additional challenges of shared spectrum, user mobility and authentication management. Not surprisingly, few organizations have the expertise, data or tools to decompose the underlying problems and interactions responsible for transient outages or performance degradations.

In this dissertation, we describe the design and implementation of an automated 802.11 wireless network diagnostic system called Shaman. First, we have de-

ployed an infrastructure of over 190 radio monitors that simultaneously capture all 802.11b and 802.11g activity in a UC San Diego Computer Science & Engineering building (1M+ cubic feet). We address the challenges posed by both the scale and ambiguity inherent in such an architecture, and explain the algorithms and inference techniques to merge all distinctive traces into single globally synchronized traces. Since the end-to-end performance of user traffic is some combination of factors across all network layers, Shaman incorporates comprehensive, cross-layer models of 802.11 network behavior and performance. These models include broadband interference at the physical layer, per-packet link layer media access delays and losses, network layer device mobility and association management, and transport layer congestion and flow control.

When users experience unsatisfactory performance at a particular time, they can query Shaman for a diagnosis. Shaman will then profile a user's traffic at that time, determine the network events that shape the performance profile, infer the causal sources of those events, and report the results to the user. Finally, we demonstrate the use of Shaman on UCSD CSE department building-wide 802.11 networks and illustrate the underlying analysis Shaman performs on real network trouble reports submitted by users of the network. We show that no one anomaly, failure, or interaction is singularly responsible for all network problems, and that a holistic analysis is necessary to cover the range of problems experienced in real networks.

Chapter 1

Introduction

Since 2000, wireless networks based on the 802.11 family of standards have become ubiquitous in the enterprise. Integral wireless interfaces — now shipping in almost 90 percent of notebook computers — combined with unlicensed spectrum and inexpensive “access points” have made untethered Internet access a reality in almost two-thirds of U.S. corporations, hospitals and college campuses [Cam05, Dal05, Del06, Gar05]. However, the reality of these deployments can be quite complex. A modest-sized office building may have hundreds of wireless users interacting with dozens of access points under varying load and environmental conditions.

Thus, there is a significant difference between installing a single wireless access point (AP) in an isolated home – effectively a simple range extender for a wired Ethernet interface – and wireless deployment throughout an enterprise. The latter may comprise hundreds of distinct APs, carefully sited and configured in accordance with an RF (radio frequency) site survey and ideally managed to minimize contention, maximize throughput and to provide the illusion of seamless coverage.

Moreover, this intricate machinery is not managed by the 802.11 protocol family itself – which, in all fairness, was never designed for the level of success it has experienced. Instead, the burden falls to the network administrator who must manage the interactions between the RF domain, link-layer dynamics, dynamic addressing and address binding, VLAN setup, as well as the myriad complexities of the wired network

itself.

1.1 Complexities in the enterprise 802.11 environment

Unfortunately, today's IT staff are poorly suited to solving such problems. In addition to the inherent complexity of diagnosing transient conditions, the complexity of the wireless environment places unique burdens on the diagnostician.

First, the radio frequency environment is itself complex. Unlike point-to-point wired networks, an 802.11 transmission must contend with signal attenuation, interference (from both narrow-band and broadband sources) and contention from competing 802.11 devices (who themselves may be mutually unaware of each other). As well, the 802.11 Media Access Control (MAC) protocol adds significant dynamism to the transmission schedule including link-layer congestion control, ARQ-based error control, dynamic rate selection and variable transmission power. Even further complicating this affair, individual vendors demonstrate significant heterogeneity in their implementation of the 802.11 "standard" and many of these impacts are not directly observable, but must be inferred.

Second, 802.11-based devices are generally designed to be mobile. Consequently, the 802.11 standard provides mechanisms for discovering, associating with and authenticating to *access points* – again with varying vendor algorithms on how best to do this. However, to manage this mobility at the network layer, most enterprise 802.11 networks employ some additional non-standard network access control, involving user authentication, dynamic network address management (typically via a combination of DHCP and ARP) and VLAN-based address mobility. All of these disparate parts are fragile – if one fails it may be sufficient to terminate a user's session and if one is overloaded it may indirectly cause transient delays across the network.

Finally, wireless networks interact via shared spectrum in ways that may not be observable directly (contention and interference) and yet can produce significant end-to-end delays or packet losses. Further complicating such analysis, the 802.11 standards

allow considerable “latitude” in the media access protocol and consequently vendors have produced a wide range of “interpretations” – many of which have significant impact on performance. Finally, the promise of seamless coverage is not a property provided by 802.11 itself. Instead, most enterprise deployments implement this property using an undeclared “layer 2.5” patched together from portions of the 802.11 protocol, combined with VLANs, ARP, DHCP and often proprietary mobility management and authentication systems. Unsurprisingly, this Rube-Goldberg contraption has its own unique failure modes.

1.2 The challenge of diagnosing enterprise 802.11 problems

Given this complexity, it is not surprising that even *simple* faults can be difficult to diagnose – let alone transient outages or service degradations. Thus, when a network manager is asked, “Why was the network flaky 10 minutes ago?” the answer is inevitably, “I’m not sure. It looks fine now.” While this problem is not unique to 802.11-based networks, these environments impose additional complexities that are unique and qualitatively harder to diagnose.

In truth, even it is technically feasible to diagnose such systems, this process is highly labor intensive and only cost effective for the most severe problems. Even then, the range of interactions and lack of visibility into their causes may stymie manual diagnosis. In one recent episode in our organization, wireless users in a new office building experienced transient, but debilitating, performance problems lasting over a year, despite extensive troubleshooting by local experts and vendor technicians. We believe we have diagnosed their problem using our system – a subtle bug in the AP vendor’s implementation – but it is easy to understand in retrospect why its discovery was challenging to find through trial and error.

It is our contention that this state of affairs is unlikely to change. Network administrators simply can’t know enough – both in the quantitative sense of examining large amounts of diagnostic data and in the qualitative sense of being an expert in

complex interactions between a wide range of network protocol layers – to solve such problems.

1.3 Contributions

Thus, we believe that such diagnosis must be automated, and that networks must eventually address transient failure without human involvement. In this dissertation, we introduce a system, called Shaman, for automating 802.11 wireless diagnoses¹. Shaman collects wireless traces from multiple distinct passive monitors and synchronizes these partial traces into a single unified global trace. It then processes distributed packet traces from both wireless and wired monitors to construct a comprehensive viewpoint of both wireless activity and dependent wired network services (e.g., DHCP). Combining this viewpoint with a causal model of protocol interactions, Shaman isolates problem root causes through a process of elimination – ranging from high-level issues (inability to transmit due to a DHCP lease timeout combined with a DHCP server failure) to low-level interactions (temporary queuing at the access point caused by broadband interference from a microwave oven).

Shaman is designed to operate in two modes: reactive and proactive. In the reactive mode, individual trouble tickets identify the MAC address and approximate time of a service problem and then system attempts to identify the cause post-mortem. By contrast, the proactive mode is designed to alert automatically on significant outages or degradations and then apply its analysis to these. To evaluate our approach we have subsumed all “help desk” functions for the production wireless network in UCSD’s Department of Computer Science and Engineering (comprising several hundred users) and have used our system to diagnose submitted “trouble tickets”. As well, we have constructed artificial problems and verified our ability to identify their underlying causes.

The main contributions are as follows:

¹The English word Shaman is derived through Russian, from the Tungus word saman meaning “one who knows”

1.3.1 Large-scale distributed network monitoring and synchronization.

Network research comes to understand the artifacts it has created slowly — by careful instrumentation, monitoring and analysis. Production 802.11 wireless networks have so far escaped the level of detailed analysis experienced on the wired network — largely because of the difficulty in monitoring the wireless environment.

Presently there are two kinds of measurement frameworks or datasets available to the research community. The first is detailed wireless traces in small office or conference settings (i.e., a few rooms). These traces typically are collected by wireless radio monitors and contain detailed MAC layer information. However, these traces are typically limited to less than 10 access points, well under a hundred clients and are taken at a single point in time. Thus, they are neither large enough to capture enterprise-scale workloads nor the complexity of wireless interactions in a large physical environment.

The second kind of wireless data available are large scale traces of campus-level wireless activity. However, because of the size of the covered network and the limitation of radio monitor ranges, these traces are limited to high-level network statistics (e.g., SNMP traces), or wired packet traces taken from access point distribution networks. Although these traces are useful in understanding high-level use patterns in wireless networks, it is difficult to perform root-cause diagnosis without fine-grained link-layer information.

To address this problem, we have deployed a large-scale distributed monitor platform in our department that reliably collects low-level traces in a 24x7 manner. We have designed and implemented a passive synchronization algorithm that can accurately synchronize over 190 simultaneous traces down to 10 microsecond granularity. To accomplish this task at scale requires predicting the impacts of individual clock skews and leveraging frames overheard at multiple radios to opportunistically resynchronize. We believe we are the first to build a monitoring platform at this scale, as well as to produce a highly synchronized trace from such diverse vantage points with reasonable storage and computation overhead.

1.3.2 Cross-layer protocol states modelings.

There are numerous opportunities for disrupting or degrading a user's connectivity in an 802.11 network. To distinguish among these, it is essential to understand the behavior (and thus the internal state) of all protocols that impact end-to-end application performance. This typically includes the 802.11 MAC, DHCP, ARP, and TCP/UDP. Moreover, these protocols frequently have cross-layer interactions that must be understood as well. For example, lost frames at the 802.11 MAC layer can be exposed to the TCP layer and thus trigger congestion-based backoff.

However, identifying the reason for such events can be quite challenging since many protocol states cannot be measured directly, but must be inferred. For example, while it is straightforward to measure how long an AP takes to forward a packet from the wired network to the wireless recipient (e.g., by simply observing the input and output frame timestamps), it is far harder to determine how much time a frame spent in the AP's transmission queue or in waiting for channel access. As we will show later in Chapter 5, these two delays can produce completely different performance diagnoses and require distinct resolutions. Shaman leverages its globally synchronized trace and observes the input and output events of a sender to model all frame forwarding process. To our knowledge, we are the first to successfully infer these 802.11 information from purely passive monitored traces.

Shaman combines a complete analysis of 802.11 MAC interactions with an analysis of TCP-layer dynamics. This capability enables Shaman to automatically infer the root-cause of performance problems of all TCP applications, including the interactive sessions or bulk transfers. We believe we are the first to systematically combine these two capabilities.

1.3.3 Real-time automated root-cause diagnosis

The last contribution of Shaman is its capability to perform fully automated root-cause diagnosis. Thus, we have developed a portfolio of protocol models for typical 802.11 clients. Shaman implements diagnosis tools that use these protocol models to

determine the root-cause of performance problems. The protocols we model are the protocols on the critical path for a typical 802.11 client to connect and use the network, including include DHCP, ARP, DNS, TCP protocols.

When a user reports a problem for his wireless connection Shaman can process the problem ticket in real-time and generate reports for both system administrators and the user. We have also implemented tools that trigger alerts when Shaman automatically detects large number of similar failures in short time period. In this case, system administrators can be quickly directed to repair key failures, but with the benefit of a complete diagnosis explaining the nature of the problem.

1.4 Overview of the dissertation

This dissertation is organized as follows: Chapter 2 presents background on 802.11 protocols and related works. Chapter 3 describes the architecture of Shaman system. The first two subsystems: monitoring and trace unification are detailed in Chapter 4 respectively. Chapter 5 presents details of the 802.11 and TCP analyzers and related diagnosis tools, followed by a summary of conclusions in Chapter 6.

Chapter 2

Background

2.1 The 802.11 wireless network protocols

This section covers the essential background knowledge of IEEE 802.11 protocols for the later parts of this dissertation. It includes access point management mechanisms, MAC channel access, b/g back-compatibility, and per-frame delivery mechanisms. The full IEEE 802.11 protocol family specification can be found in [Com99].

The 802.11 protocol, also known as Wi-Fi, evolved from the Lucent WaveLAN protocol developed in late 90s. The WaveLAN protocol was intended to serve as a simple “wireless extension” of the Ethernet protocol, and the two protocols share many of design elements in the medium access (MAC) layer. 802.11 is designed to be used in indoor use, e.g., residential homes or office buildings. The effective radio range is about 100 - 50 meters for typical deployments.

2.1.1 Network management and configurations

In the “infrastructure mode” of 802.11, clients connect to access points (APs) connected to the wired network, called “the distribution system”, typically running the Ethernet II protocol. The AP is essentially an Ethernet bridge that forwards packets in and out of the wired network. Instead of having centralized management control, in 802.11 all the nodes (clients and APs) coordinate with each other in a distributed fashion

to gain access to the wired network and access the physical channel.

To enable clients to locate an available wireless network, the APs periodically broadcast beacons to announce their network identifications (SSIDs), or clients can actively send probe requests to the APs. Then the clients can send an association request to join the network announced by a particular AP. If the AP grants the association, then the client can start sending normal application traffic through the AP to the wired network. Typically the client performs regular Ethernet access operations like getting a DHCP address, ARPing the gateway, and running TCP applications. Thus after the association, the AP creates an illusion of the client plugged into an Ethernet cable.

In enterprise 802.11 networks deployed in large office buildings, the most common configuration is to deploy multiple homogeneous APs that announce the same SSID. The APs are connected in the same subnet, through VLAN, to the wired infrastructure. Usually the wireless VLAN has a designated DHCP server that serves the wireless subnet IP designations. Many enterprises also deploy authentication servers to control access to the internal enterprise network.

2.1.2 MAC channel access

The 802.11 media access control protocol is a CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) variant that uses “virtual carrier sense” to support an RTS/CTS (Request to send)/(Clear to send) capability and to protect multi-frame exchanges. When a node wishes to send, it first validates that the channel is clear. If the channel stays idle for a set period of time (DIFS) it transmits. Otherwise, it selects a random backoff time in $(0, N]$, and tries again. Since an 802.11 radio is a half-duplex transceiver, the sender requires acknowledgment from the receiver to respond immediately with an ACK frame for all unicast transmissions.

If the sender does not receive an ACK within a preset timeout, it doubles N , calculates a new (likely longer) backoff time, and schedules a retransmission (retransmissions are indicated with a special bit in each frame header). Thus 802.11 implicitly assumes a failed transmission (not hearing an ACK) indicates frame collision and al-

ways performs a back-off. To avoid a node seizing the channel, which may lead to the Ethernet “capture effect” [KR94], the sender must perform mandatory back-off after sending all the retries of a frame, regardless of whether the delivery is successful or not.

In the virtual carrier sense, each frame carries a “duration” field that indicates the number of microseconds needed to complete the transaction (including any acknowledgments that need to be sent) and each node will defer transmission until this time has passed. Special RTS and CTS frames are optionally used to ensure that any “hidden terminal” nodes will hear the reservation request.

Frames are addressed using 48-bit IEEE MAC addresses, although some frames (such as ACK, CTS and RTS) only specify the transmitter or receiver. Frames from the same transmitter are distinguished using a 12-bit monotonically increasing sequence number carried in each DATA or MANAGEMENT frame. Special management frames (BEACON and PROBE) are used to discover the presence and capabilities of access points, while others (ASSOCIATION and AUTHENTICATION) are used to specifically connect a client to an access point.

2.1.3 PHY implementations

The majority of 802.11 networks operate in the 2.4GHz (ISM) band. This band is public and shared by many other home/office appliances such as microwave ovens and cordless phones. These devices typically do not understand 802.11 and may severely degrade 802.11 performance because 802.11 protocol requires participants to coordinate channel access. We discuss the impact of broadband interference further in Chapter 5.

802.11 supports a wide range of physical-layer implementations — the most popular being 802.11b (CCK modulation with coded rates up to 11 Mbps) and 802.11g (OFDM, coded up to 54 Mbps). Each client is responsible for choosing the rate to transmit each frame and this choice is encoded in the PLCP header at a “slow” rate (1-2 Mbps for 802.11b, 6 Mbps for 802.11g). However, “legacy” 802.11b radios are unable to decode the OFDM encoding of an 802.11g frame and can incorrectly sense

the medium as idle. To address this problem, 802.11g access points determine if they have any 802.11b stations as clients. If so, they enable “802.11g protection mode” in which each 802.11g frame is preceded by a low-rate CCK-coded CTS frame (CTS-to-self) that reserves the channel for the time needed to complete the 802.11g transaction.

2.2 Wireless network measurements and characterizations

From the mid-90s, a progression of wireless network measurement efforts have provided insight into the behavior, performance, and reliability of wireless LAN technologies. Starting with small studies focused on low-level channel behavior between pairs of nodes [DR92, ES96, NKNS96] the field has expanded to cover a range of more abstract characteristics (including application workloads, user session duration, user mobility, increasingly larger environments, etc.) over ever larger environments (including university campuses [KE02, HKA04, HCP05, MV05, SB04, TB00, YYA04], industrial factories [WKHW02], corporate networks [BC03], and conference and professional meetings [BVBR02, JRABR05, RBRA04, RRM⁺05]). However, as measurement scale has increased, methodological challenges have led most researchers to treat wireless networks as a black box and instead base their analyses on wired distribution network traffic and polled SNMP management data from APs. As a result, existing measurement efforts have extensively characterized *what* user behavior and network performance wireless LANs provide, but have provided little insight into *why* applications and users experience such behavior and performance.

In 2004, researchers began addressing this question by extending wireless network measurement to passively capture and analyze link-level characteristics as well. Yeo *et al.* [YYA04, YYHA05] were the first to explore the feasibility of using separate monitors for passive wireless network measurement using synthetic experiments on an isolated 802.11 network. They use beacon frames to merge traces of a single flow observed from three wireless monitors, and demonstrate the utility of merging observations to improve monitoring accuracy. Jardosh *et al.* [JRABR05] analyze the link-level

behavior of traffic from a large IETF meeting using three monitors capturing traffic on orthogonal channels. They characterize and correlate retransmissions, frame size, and rate adaptation with reliability. Finally, studies by Rodrig *et al.* and Mahajan *et al.* share a number of the goals of our work [MRWZ06, RRM⁺05]. They use five distributed wireless monitors to capture network events in a large conference venue. Using this trace data they analyze various performance characteristics of the 802.11 MAC protocol. Their work is distinguished by their learning approach for automatically characterizing protocol interactions, while ours has focused on the problems of large-scale online monitoring and complete multi-layer reconstruction.

2.3 Critical path analysis of networking problems

Other techniques infer more detailed network events and characteristics, such as link-layer loss and the transmitters of packets lacking MAC addresses [CPWZ07, MRWZ06], misbehaving and heterogeneous devices [GBG06, CPWZ07], root causes of physical-layer anomalies [SDG⁺06], and regions of poor coverage [CPWZ07]. We greatly expand upon these detailed efforts and present techniques to infer critical path delays [BC00] of media access for every packet, such as AP queuing delay and media contention (mandatory and regular backoff), as well as techniques that infer management delays for supporting intermittent and mobile devices for every user.

To infer critical path delays for wireless transmissions, we develop a detailed model of 802.11 media access in 5. Numerous models have been developed previously to estimate various aspects of 802.11 performance, such as the overall network capacity as governed by the 802.11 protocol [CBV05], the maximum throughput of a flow in an 802.11 network [JPS04], and the saturation throughput and expected access delay of contending nodes [KSM05]. Such models are typically analytic. To make analysis tractable, they explicitly make simplifying assumptions such as absence of transmission errors, uniform transmission rates and packet sizes, infinite node demand and steady contention for media access, uniformly random probability of collisions and interfer-

ence, etc. As a result, these models may be useful for understanding the limits of 802.11 performance under idealized conditions, but omit analysis of important aspects of real networks that we infer with our model: the magnifying effects of bursty traffic that averages and expected values conceal, and the complexities of workload, protocol, and environment that lead to correlated and unexpected interactions.

2.4 Wireless network diagnosis systems

Several research systems are closely related to the goals of this dissertation. We compare each system with the Shaman system individual as follows:

The Client Conduit (CC) is the first research wireless diagnosis system [ABCQ04] that shares a common goal with Shaman. But the approach is different. Instead of deploying passive monitors, CC installed software in the AP, clients, and a back-end server. The client and AP software relays probing or diagnosis information using the 802.11 Ad-hoc mode to the back-end diagnostic server. The server then diagnoses for connection or performance problems. CC diagnoses TCP performance problem using high-level packet statistics like RTT and DUP-ACKs, whereas Shaman performs more low-level diagnosis of the root cause of TCP connection problems.

DAIR [BCP⁺06, CPWZ07] is another system that helps system administrators diagnose WiFi problems. DAIR and Shaman use similar approaches, distributed wireless monitors, for monitoring detailed wireless events in an enterprise network — DAIR uses wireless USB dongles attached to standard desktop machines, while Shaman uses a monitoring infrastructure of specific embedded monitor boxes. DAIR applications install trackers on the desktop machines to trace information of interest and store it in a central database; applications (inference engines) then query this database to perform analyses. DAIR uses a sophisticated location algorithm to estimate client locations accurately, and narrow diagnoses relative to location; in contrast, Shaman uses a merged trace of global activity across the entire network. Unlike Shaman, though, DAIR does not perform detailed PHY or link layer loss or delay analyses due to lack of low-level

traces. Our goals are similar in that we develop analyses to aid network management, but our approach is to base analysis on a global understanding of network behavior across all protocol layers.

Shaman shares some goals with the WiFiProfiler [CPZ06], which also helps users troubleshoot wireless connectivity problems. The two systems take different approaches, however. WiFiProfiler relies upon peer diagnosis among clients without the involvement of system administrators, while Shaman relies upon third-party monitoring and inference. WiFiProfiler installs custom software on the client to collect detailed network stack statistics, such as beacon losses and queue length, as well as OS and driver details. It then exchanges this information with peers to diagnose connectivity problems. The client can then determine if it has an association problem, DHCP problem, or TCP problem.

In WiFiProfiler, due to the local knowledge exchanged from the peers, however, the diagnosis is restricted to determining relatively high-level causes. For example, the client TCP diagnosis can indicate high TCP loss rates, but not the cause of the losses. The advantage of WiFiProfiler is that it has no infrastructure requirements and is best suited for ad-hoc first-step diagnosis. On the other hand, the diagnosis is limited to high level behavior and users may raise security/privacy concerns over installing custom software to exchange detailed OS information with other users. Overall, Shaman and WiFiProfiler are complementary to each other because each obtains additional knowledge about the wireless network that cannot be perfectly inferred.

The MOJO system develops tools and techniques to identify the root causes of physical-layer performance anomalies, such as broadband interference and the capture effect [SDG⁺06]. While we are interested in the effects of physical-layer issues, we identify them as just one cause among many across the interacting protocol layers.

Besides research systems, various commercial products [aira, airb, kis] and research systems have been developed to monitor and diagnose 802.11 wireless networks. Research systems in particular have evolved from developing infrastructure for performing distributed wireless monitoring and demonstrating the analysis capabilities

of such platforms [HRABR04, YYA04, RBRA04] to systems for diagnosing problems in wireless networks [ABCQ04, BCP⁺06, CPWZ07, SDG⁺06, QBRZ06, RHA04].

2.5 Summary

Overall, our work extends previous efforts in wireless network monitoring in terms of scale, performance, methodology, and analysis. Whereas previous efforts have used a small handful of monitors [RRM⁺05, YYHA05], our measurement platform uses over 190 monitors distributed throughout four floors of a 150,000-square-foot building for extensive spatial and channel coverage. Tracing at such scale, however, presents new methodological challenges, such as globally synchronizing events in time across subsets of monitors as well as across channels; previous efforts either focus on separate channels [JRABR05], do not merge traces among monitors [RRM⁺05], or merge only a small number of traces *offline* using globally observed events [MRWZ06, YYHA05]. Such extensive on-line monitoring capabilities also presents new opportunities for analysis, in particular the ability to observe a large wireless network from a global perspective. Finally, our ability to unify this global view among physical, datalink, network and transport layers creates the opportunity to study cross-layer interactions directly.

Chapter 3

Shaman System Architecture

Shaman consists of four main components. Figure 3.1 illustrates the relationship of these components and how data flows among them. This chapter gives a brief walk-through of all the components.

3.1 Monitor infrastructure

Shaman monitors the enterprise wireless network from two different vantage points, the wireless monitors and the wired distribution network. Each wireless monitor passively monitors the channel and records all wireless events including frames with CRC and PHY errors. The monitors capture the first 120 bytes of most received frames; for DHCP frames, we capture 400 bytes to include the necessary DHCP headers. The monitor compresses and streams the traces back to a central storage server.

Shaman also records wireless network traffic as it traverses the wired distribution network. A SPAN port on the router for the building distribution network forwards all packets to a wired tracing machine, which also has a wireless monitor. Tracing packets on the wired distribution network is necessary for inferring detailed media access delays (chapter 5.4); in particular, we need to infer when wired packets arrive at the APs as the basis for estimating queuing within the AP. We attach an additional wireless monitor directly to the wired monitor to synchronize across the the wired and wireless time domains.

Figure 3.1: The Shaman system architecture.

3.2 Synchronization and preprocessing

Shaman merges and time synchronizes all of the traces from the wireless monitors into a single global unified trace. Shaman implements synchronization based on Jigsaw's algorithms [CBB⁺06]. It merges frame transmissions observed by multiple monitors into a single frame, and timestamps each frame according to a global virtual clock. For example, if three monitors receive a unicast DATA frame transmitted by an AP to a client, Shaman will identify these frames as equivalent and create a frame representing that single transmission.

We have augmented Jigsaw's synchronization with multiple enhancements to better support its use in a diagnostic system. First, we track the offset between the synchronized virtual clock and the real-time clock of the wired trace monitor, and timestamp frames according to the real-time clock. Since our wired trace monitor also has a wireless radio that collects wireless traces, we slave the synchronized virtual clock to the real-time clock by putting real-time clock timestamps in that particular wireless radio's traces. Using this timestamp serves two purposes. One, it synchronizes the time domains of the wireless and wired traces. Two, it enables the diagnostic system to easily correlate times specified in user queries with timestamps in the frame trace. Second, we have tuned Shaman to synchronize and compress the traces in real time with a relatively

short delay. Every minute, the synchronization process takes 3-15 second on a Pentium4 2.4GHz machine which enables Shaman to perform real-time analysis.

3.2.1 Protocol states inference

The output of synchronization is a continuous stream of frames in a custom *jcap* format. A *jcap* trace has the format of a gzipped pcap file that combines a *jcap* header with every frame. The *jcap* header includes the transmission rate, preamble length, etc., in a 24-byte record for each frame. A record in the *jcap* trace is 170 bytes on average, resulting in only a moderate trace data rate even for large monitored networks.

The frame sanitizer examines frames for anomalous or inconsistent fields, and corrects or drops them to ensure that all frames are valid before further processing. Inconsistencies are typically the result of buggy 802.11 firmware implementations. For example, we regularly see frames with anomalous durations (e.g., NAV fields with 0xFFFF) and packets with inconsistent types and lengths (e.g., wireless security Webcams in our building transmit ACKs with impossibly large packet lengths). We mark these frames and adjust frames whose fields we can correct (e.g., by calculating the correct value for the NAV field based on the frame length and rate). A later analysis module can then process these frames without concern.

The station module tracks state related to a station's association with the network as a function of time. This state includes the preamble mode (short or long), slot time (short or long), power save mode, the use of 802.11g protection mode and RTS/CTS, etc. The station module maintains this state directly, by tracking the parameters advertised in AP beacons and client scan probe frames, and indirectly, based on the timing of successive frame transmissions (e.g., sufficiently fast ACK responses preclude long preambles).

The station module provides reconstructs individual link-layer conversations using a *frame exchange constructor*. The constructor starts by identifying all *transmission attempts*. A transmission attempt usually consists of control frames, a data frame, and one ACK from the receiver (e.g., RTS-CTS-DATA-ACK is a common frame ex-

change pattern for 802.11g clients). The constructor then groups transmission attempts into complete *frame exchanges*. Since 802.11 implements ARQ for unicast frames, a frame exchange may involve multiple distinct transmission attempts. Normally it is sufficient to simply group nearby transmission attempts that share the same frame sequence number. But since the wireless monitors can fail to capture some frames, the frame exchange constructor performs contextual inference to compensate for any omissions [CBB⁺06].

Typically, a frame exchange represents a packet delivered either to or from the wired distribution network. Thus, successful frame exchanges usually also have a counterpart in the wired trace. The job of the cross referencer is to match the packets in the wired trace (Ethernet II frames) to the frame exchanges on the wireless side with identical data content. The output of the cross reference module is the matched frame exchanges which are used to drive various analysis modules.

3.3 Critical path diagnosis tools

Shaman provides two diagnostic tools, one designed for users to invoke on demand about wireless problems that they are experiencing and a second designed to alert network administrators about pervasive network problems.

The goal of the Shaman user diagnostic tool is to answer queries on demand from users about performance problems that they are experiencing with the enterprise wireless network. When a user submits a ticket through Shaman's web interface, the tool executes on a backend server and identifies the particular user's trace by the provided MAC address. Shaman then checks the protocol states of the hosts on the critical path, including 802.11 associations and scans, DHCP, ARPs, and TCP protocols.

The goal of the Shaman network alert tool is to pro-actively report serious pervasive problems to network administrators. Serious pervasive problems are those that simultaneously affect multiple clients at one or more APs and require the intervention of a network administrator to correct. Typically these problems are due to failures

of critical network components (DHCP or DNS servers, routers, wireless management gateways, etc.), or persistent performance issues.

3.4 Acknowledgement

Chapter 3, in part, is a reprint of the material as it appears in the SIGCOMM Conference, 2006, Cheng, Yu-Chung; Bellardo, John; Benko, Peter; Snoeren, Alex C.; Voelker, Geoffrey M.; Savage, Stefan. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, is a reprint of the material as it appears in the SIGCOMM Conference, 2007, Cheng, Yu-Chung; Afanasyev, Mikhail; Verkaik, Patrick; Benko, Peter; Chiang, Jennifer; Snoeren, Alex C.; Voelker, Geoffrey M.; Savage, Stefan. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Network Monitoring and Trace Synchronization

This chapter describes the first two components of the Shaman system. A combination of wireless and wired network monitors collect network traffic in real-time as input into the system. Then a series of synchronization and preprocessing steps synchronize and merge the traces, normalize frames, track station association status, and reconstruct frame exchanges from individual transmission attempts.

4.1 Monitor infrastructure

Any data analysis is ultimately predicated on the quantity, quality and precision of data that can be collected. While we believe that our analysis techniques are mostly generic, many of our design decisions *have* been informed by the capabilities of our platform as well as the unique problems presented by its scale. For example, our approach to clock synchronization was driven by the need to merge data from 193 simultaneous traces, spanning a wide spatial and frequency range. In a smaller-scale environment a far simpler approach would have sufficed. Thus, to better motivate our constraints and opportunities, we use this section to describe our monitoring environment and the hardware/software platform we have built to produce the raw traces for

Figure 4.1: UCSD CSE building floor plan. This building comprises roughly 150,000 square feet over four floors and a basement. Circles indicate wireless sensor pods, and triangles indicate production access points.

our analysis.

4.1.1 Environment

All of our measurement work takes place within the UCSD Computer Science and Engineering department building, a large four-story structure shown in Figure 4.1. The building houses over 500 faculty, students and staff members within roughly 150,000 square feet with a total interior volume well over 1 million cubic feet. Avaya AP-8 access points (shown as triangles) provide production wireless service, configured

for both 802.11b and 802.11g service.

Between and among these production APs we have deployed a constellation of 48 wireless sensor pods (shown as pairs of circles). Our monitoring platform does not cover the left wing of the first floor, which is not under our administrative control. Each pod in turn comprises two monitors for four independent radios, allowing for simultaneous monitoring at four distinct center frequencies — including all “non-overlapping” channels (1, 6 and 11) typically used in 802.11b/g deployments. In addition, we have one more radio installed in our back-end server for synchronizing the universal time standard and our virtual global clock (see Section 4.2.2). Thus in total we have 193 radios monitoring the network. The density of deployment, combined with this multi-channel capability, provides a “best case” scenario for capturing global behavior. We are unaware of any production wireless network monitored at similar scale.

4.1.2 Hardware

Soekris monitor box. Concretely, each sensor pod consists of a pair of monitors set a meter apart. This organization provides sufficient antenna separation for active measurement experiments, while still being proximate enough to abstract both monitors as a single vantage point for passive monitoring. Each monitor consists of a modified Soekris Engineering net4826 system board, and couples a 266-MHz AMD Geode CPU, 128 MB of DRAM, 64 MB of flash RAM, a 100-Mbps Ethernet interface, and two Wistron CM9 miniPCI 802.11a/b/g interfaces based on the Atheros 5004 chipset. Each wireless interface is connected, via shielded cable, to a separate external omnidirectional “rubber duck” antenna mounted six inches apart on an aluminum enclosure. The antennas provide a signal gain of 2–3 dBi at 2.4 GHz. Each monitor receives wired connectivity and power through a port on an HP 2626-PWR switch (seven in total). We refer the interested readers for the most detailed monitor hardware constructions to [Bel06].

Backend server. All the monitors feeds the traces via NFS continuously to a back-end server. The server has double AMD Opteron 2.2Ghz CPUs, 4GB ram, and two

2.6 terabytes RAID-5 disk arrays. The server runs the Shaman software continuously including the synchronization, analysis, and diagnosis tools. The server also runs several routine services for all our monitors like DHCP, PoE controllers, etc.

4.1.3 Software

Each monitor runs a version of Pebble Linux, using the [mad03] driver to drive the Atheros-based wireless interfaces. We have made significant modifications to the driver to support additional transparency to the physical layer and to improve capture efficiency.

Madwifi driver modifications

While the standard madwifi driver only delivers valid 802.11 frames (even in so-called “monitor mode”), our version captures all available physical layer events, including corrupted frames and physical errors. Atheros hardware uses a $1\ \mu\text{s}$ resolution clock to timestamp each packet as it is received. Our driver slaves this timestamp facility to the clock of a single (first) radio, thereby recording frames at both radios using the same time reference.

Since the monitor hardware has limited processing capability and memory, extreme packet bursts, e.g., physical errors bursts caused by microwave ovens, can cause receiver livelock in the wireless monitor, deplete its memory and trigger the watchdog to reboot the monitor. This would leave 2 minutes of trace holes in our raw wireless traces. We solve this problem by disabling the the radios’ interrupts for one second when monitor memory goes lower than a threshold. To further reduce the kernel-user copy overhead, our driver batch the frames into 16KB buffer before uploading to the user-land capture software.

Jigdump

A specialized user-level application called *jigdump* manages data capture. Each monitor executes two *jigdump* processes, one per radio, that are responsible for

putting the wireless interface into monitor mode, “pulling” physical event records from the kernel, and then transferring this data via NFS to a central repository. Jigdump reads data records 16 KB at a time via a standard `PF_PACKET` socket, compresses them using the LZO algorithm to minimize storage and I/O overhead (the two bottlenecks on our monitor platform) and generates a meta-data index record to facilitate subsequent accesses. Data and meta-data are written to separate files via NFS.

4.1.4 Coverage

A fundamental challenge with distributed wireless monitoring is obtaining effective coverage of all network transmissions. Monitors must be carefully placed to maximize the probability of “hearing” all clients and APs. Even so, the vantage point of a monitor is distinct from those it is monitoring and thus some network transmissions may go unobserved due to attenuation, noise, interference, etc. In this section we present two experiments that empirically evaluate the coverage of our monitoring platform, and a third synthetic experiment to evaluate the sensitivity of these results to the number of radio monitors used.

Originally we only deployed 39 sensor pods (156 radios) in the hallways of the first to fourth floors of the building, excluding the basement. The following coverage experiments are based on this settings. To establish the coverage of our link-layer monitoring capability, we performed a controlled experiment comparing our results against an “oracle”. Using a wireless laptop, we generated a network workload at various locations throughout the building. The workload was a combination of Web browsing on the Internet, interactive `ssh` sessions to wired hosts, and `scp` copies of large files (producing both short and long flows as well as small and large packets). We generated this workload at three locations in each wing of each floor. During the experiment the laptop recorded all link-layer events it generated and observed from its associated APs. Conversely, we used the monitoring platform to simultaneously observe the laptop’s communications. Comparing these two versions of events, the monitoring platform observed 95% of all link-level events generated by the laptop. The coverage in this experiment

Figure 4.2: Coverage of frames transmitted by clients and APs.

is consistent with smaller-scale studies using similar wireless monitoring methodology: [JRABR05] reports a coverage of 80–97%, [RRM⁺05] reports 90%, and [YYHA05] reports 97%.

Next, we compared the frame exchanges captured in a day-long trace of the wireless network (described in more detail in the next section) with a second trace of the same traffic captured on the wired distribution network. We restricted the comparison to the set of flows that could be possibly observed at both vantage points; the monitor for the wired network, for example, does not observe traffic sent from one wireless host to another. For every packet in every flow in the wired trace that would result in a unicast DATA packet on the wireless network, we checked to see if the packet also appeared in the wireless trace. Overall, the coverage is excellent. For the 10 million unicast packets in the wired trace, 97% of those packets also appear in the wireless trace. This high coverage is particularly encouraging since the trace includes distant clients connected to the building APs from the administrative wing on the first floor, locations lacking monitors.

Figure 4.2 shows the results of this experiment in more detail. Across all stations in the wireless access network, it shows the percentage of unicast DATA frames transmitted by the stations that appear in the wired trace that also appear in the wireless trace. It also separates the stations into clients and APs. The graph shows that, for many stations (46% of clients, 40% of APs), the monitoring platform captured all of their transmitted frames. And, for most stations (78% of clients, 94% of APs), the platform captured over 95% of their transmitted frames. The clients with substantial missing frames were located in rooms that consistently lack good coverage by the monitoring platform. We also see that coverage differs depending on station type: the monitoring platform captures a higher fraction of packets transmitted by APs than by clients because our sensor pods are purposely placed in AP proximity, while wireless clients are dispersed throughout the building.

Finally, we evaluate the extent to which our deployment of sensor pods is necessary to achieve good coverage in our building. In this experiment, we successively reduce the number of sensor pods that contribute to the unified wireless trace. We then determine the resulting coverage of frames that appear in the wired trace that are captured in the reduced wireless trace. To reduce processing time for each configuration, we calculate coverage of traffic generated between 11am and 1pm—the peak hours of wireless traffic in our building. We manually choose pods to remove based upon “visual redundancy” of pod locations in the building: we remove pods at locations that appear to have overlapping coverage by other pods as seen in building floor plans. Rather than determine an offline optimal pod selection that maximizes coverage knowing the trace contents, this reduction method reflects the level of knowledge available when placing pods for the first time (indeed, it is precisely the algorithm we used for building our monitor platform—simply with fewer pods).

Figure 4.3 shows the sensitivity of coverage to the number of sensor pods. A pair of bars shows the coverage of unicast frames transmitted by APs and clients that appear both in the wired trace and in the resulting wireless trace for a particular pod configuration. Each pair of bars corresponds to three different monitoring configurations

Figure 4.3: Coverage of frames transmitted by APs and clients for various configurations of sensor pods in the building.

of all 39 pods (156 radios), 30 pods (120 radios), and 20 pods (80 radios); reducing to 10 pods creates partitions the synchronization bootstrap trees, preventing complete trace unification. Coverage of AP frames remains good (94%) even operating a third of the original pods due to the ability to have few obstructions between pods and APs (since both are typically mounted in corridors). Coverage of client frames, however, drops dramatically (from 92% to 71% to 68%) as we reduce the number of sensor pods.

From these results, we conclude that we need the full set of pods to achieve good coverage for most of the client transmissions in our building; but we have identified locations where our monitors do not cover well, like our basement labs and some corner offices. After these experiments, we added another 18 monitors to improve our coverage. In our current settings, we can cover 99% of all the transmissions. Based upon the coverage measured in these experiments, we conclude that the monitoring platform provides sufficient coverage to perform detailed analyses of traces captured using the platform.

4.1.5 Monitoring performance and overhead

Our monitors and Jigdump software are very stable after we implemented the receiver livelock relief mechanisms. Normally we only experience 1-2 reboots in a week across all monitors. The aggregate NFS traffic for all traces feeds averages 8Mbps and is about 10 times higher than our production wireless traffic. The raw traces average 80GB and 40GB of storage daily on weekdays and weekends respectively.

4.2 Trace unification and synchronization

Each individual trace from our monitor platform represents a particular local vantage point on wireless activity. To construct a global viewpoint it is necessary to combine traces from all the radios into a single coherent description.

In this chapter, we describe this synchronization subsystem of the Shaman system. This merging procedure must satisfy three key requirements:

1. *Unification.* Several radios may observe a particular frame which therefore appears in multiple traces. It is important that we identify these “instances” as corresponding to a single physical 802.11 transmission. In some cases a received frame may not even be a perfect instance (*e.g.*, due to corruption or truncation in the receiver) yet it should still be associated with the same transmission.
2. *Synchronization.* While the monitors timestamp each frame in each trace, the radio clocks can vary significantly without explicit clock synchronization. To place these frames in proper order, it is necessary to synchronize all frames to a common reference time. Merely capturing the logical order is not sufficient for performing fine-grained 802.11 timing analyses, such as inferring interference between simultaneous transmissions. Such studies require all frames to be synchronized to at least the precision of a physical layer “slot time” (10 μ s for 802.11b and 802.11g).
3. *Efficiency.* To permit online applications, trace merging should execute faster than

real-time and scale well as a function of the number of radios. Thus, we prefer an algorithm that can merge traces in a single pass over the data.

The goal of synchronization is to provide a globally synchronized trace of the entire wireless network with timestamps close to $10\ \mu\text{s}$ precision. The challenges are the radios in our monitor platform are not synchronized and we may encounter large relative skews between pairs of 193 radios. While our monitors run the Network Time Protocol (NTP) to synchronize the system wall clocks, NTP only provides milliseconds level accuracy. For this reason trace frames are timestamped by the hardware clock in the radio, not the monitor wall clock.

Our approach, similar to Yeo *et al.*'s framework [YYA04], exploits the broadcast nature of wireless. Since wireless is fundamentally a broadcast channel, multiple in-range receivers can potentially record each transmission. Moreover, in an indoor environment, propagation delay is effectively instantaneous — less than $1\ \mu\text{s}$ to cover 500 meters at 2.4 GHz. Consequently, we can treat the time at which a given frame is received by multiple radios as a simultaneous event for all potential interactions. Thus, we can use frames heard by multiple radios as a common reference point to synchronize the clocks at each radios and globally order subsequent events between traces.

Subsequently, we can use these reference frames to calculate global timestamps for subsequent events *within* each trace by using local clocks to place the remaining frames in relation to reference frames. Finally, we can unify identical instances with the same timestamps, thereby creating a single global trace. In the remainder of this chapter we describe Shaman's synchronization and unification algorithms.

Our synchronization approach is inspired by Elson *et al.*'s RBS protocol for sensor networks, which shares many of the same assumptions [EGE02]. The two algorithms, however, diverge in implementation due to the differing demands of their applications: Shaman is opportunistic in finding time references yet permits a centralized implementation, while RBS mandates reference broadcasts but requires a distributed implementation. Most importantly, RBS provides relative time synchronization between pairs of sensors, while Shaman must accurately synchronize all traces to a single global

clock. Accomplishing this task involves two phases: bootstrapping the synchronization algorithm to instantiate a single universal time standard across all radios, and then maintaining this standard during frame unification.

4.2.1 Bootstrap synchronization

Shaman bootstraps synchronization by finding reference points to synchronize the radios of a set of individual monitors, and then synchronizes among sets until it establishes a single coordinated time standard called “Shaman Universal Time” (SUT). Note that SUT is virtual, and we will describe how to calibrate SUT into real-life Coordinated Universal Time (UTC) later in Section 4.2.2. We begin with the assumption that all radio clocks run at the same rate, and then consider skew.

Let r_i denote the i th radio and let T_i represent the offset between its clock and SUT. Let s_k denote the k th reference 802.11 frame used to synchronize radios and let E_k be the set of pairs $\langle r_i, s_k \rangle$ such that radio r_i receives frame s_k . We denote $\langle r_i, s_k \rangle$ as an “instance” of frame s_k . Essentially, an instance is a packet recorded in our radio trace file from our Jigdump software. Finally, let y_{ik} denote the reading of r_i ’s clock when it received s_k (defined if and only if $\langle r_i, s_k \rangle$ is in E_k). We refer to y_{ik} as the “local timestamp” of instance $\langle r_i, s_k \rangle$ recorded at radio r_i . Thus, when s_k has been received at radio r_i , the SUT timestamp can be defined as

$$U_k = y_{ik} + T_i.$$

To bootstrap synchronization, Shaman must find an assignment of T_i for each radio. Once the offset T_i is available, Shaman can determine the SUT timestamp of each frame s_k .

Ideally Shaman could locate a single reference frame s_k where E_k contains every radio. Then y_{1k} could be picked arbitrarily to represent the initial SUT. Unfortunately, we cannot depend on such events in a large deployment since signal strength attenuates with distance and no single frame likely covers an entire building. Moreover, real deployments use multiple channels and a frame transmitted on one channel may

never be heard by a monitor on another.

To overcome this problem, we synchronize transitively via overlapping subsets of radios that are each synchronized with each other. For example, suppose radio r_1 and r_3 are too far apart to share any reference frames, but each shares distinct reference frames with an intermediate radio r_2 . If s_A is a reference frame received only by r_1 and r_2 , and s_B is a reference frame only received by r_2 and r_3 , then $y_{1,A} + T_1 = U_A = y_{2,A} + T_2$, and $y_{2,B} + T_2 = U_B = y_{3,B} + T_3$. Then $T_3 = y_{1,A} - y_{2,A} + y_{2,B} - y_{3,B} + T_1$. The more densely the radio deployment, the more such transitive paths between r_1 and r_3 are likely to exist. However, to maximize the likelihood that T_i s are globally consistent — *i.e.*, $(T_j - T_i)$ plus $(T_k - T_j)$ equals $(T_k - T_i)$ — we try to maximize the overlap between paths by minimizing the number of distinct reference frames.

Our protocol works as follows. Shaman examines the first second of data from each trace. In this case, “the first second” refers to UTC as measured by the system clock on each monitor. Each monitor maintains their system clock within milliseconds using the NTP protocol and every instance carries an UTC timestamp in addition to y_{ik} . For every s_k that appears in the first second of the trace, Shaman identifies its instances and adds them into E_k .

This instance identification is done by comparing frame content (including sender/receiver MAC addresses, etc.) which requires 802.11 frames sent in a one-second interval to be uniquely identifiable. While 802.11 data and management frames carry a 12-bit sequence number that monotonically increase, not all 802.11 frames are good references for synchronization. For example, ACK frames to the same destination are always identical, some stations always use zero sequence numbers on probe frames, and frame retransmissions cannot be distinguished from one another. Thus, Shaman only uses “unique” frames for all synchronization activities. Generally, these are data or management frames that do not have the retransmit bit set. Shaman also excludes some types of frames from particular vendors that do not follow the 802.11 specs to use unique sequence numbers. For instance, some 802.11 implementations incorrectly retransmit data *without* the retransmit bit set, but thankfully this is rare. Although we

skip some vendors' frames to bootstrap synchronization, we still analyze their traces.

For every radio trace, Shaman picks the set E_k that contains the maximum number of radios and adds it into the synchronization set G . Shaman stops filling G when G contains an instance of each radio. Then, for each radio r_i , Shaman performs a breadth first search in G to reach r_1 . Notice there may exist multiple paths to reference the time from r_1 to r_i . Karp *et al.* [KEES03] have discussed ways of picking the optimal paths for a similar problem, but we have found that most paths from r_1 to r_i are precise enough in practice ($\pm 10 \mu\text{s}$).

There usually exists at least one path between any arbitrary two radios on the same channel (if not, the original one-second window could be advanced or more sets E_k added to G , but we seldom need to do this). However, Shaman is unlikely to find a path between radios on strongly disjoint channels, e.g., radios monitoring on channel 1 and channel 11. To fully synchronize across channels we exploit the fact that our monitors use a single clock to timestamp frames received on both of their radios, i.e., the second radio timestamps frames with the first radio's clock. Thus, in this particular context local timestamps for frames on one channel can be directly related to timestamps on another — effectively bridging a path between them.

4.2.2 Frame unification

After bootstrap synchronization, Shaman obtains the offsets T_i for every radio r_i to the SUT. Therefore Shaman can compute the SUT timestamp of every frame recorded in the trace by adding T_i to the local timestamps. In other words, all radios have agreed upon a universal time SUT. In the rest of this chapter, unless stated specifically, time refers to the SUT time standard.

Shaman processes all traces in time order and unifies instances of 802.11 frames from multiple radios into a frame that represents an actual 802.11 frame transmitted in the network. Each frame holds a SUT timestamp, the full contents of the frame and the identity of the radios that heard each instance. Figure 4.4 provides an example of this source data as it is being unified. As part of the unification process, Shaman also

Figure 4.4: Visualization of synchronized trace. Time appears on the x-axis in μs and individual radios on the y-axis. At 400 μs , six radios receive a data frame sent by a client and five of them, except p4450, receive the corresponding ack

resynchronizes the clocks between each trace to account for skew and drift. We describe the evolution of the algorithm below.

Basic unification

For each radio trace Shaman maintains an instance queue sorted in time order. The simplest unification approach is to linearly scan the head of all radio queues and group the instances with the same timestamps and contents. More concretely, Shaman will select the first valid frame (*i.e.*, FCS was successful) as the representative instance and then perform content comparisons to find instances among the candidates whose SUT timestamps are within a “search window”. In our environment, the search window is 500 μs for unique frames and 200 μs otherwise.

To quickly prune false negatives, Shaman compares frame length, rate, and FCS fields first and short-circuits the comparison on failure. For partially received or corrupted frames, Shaman cannot perform a full content comparison and simply matches

Figure 4.5: Frame unification among five radios. Shaman (a) uses a search window to identify candidate instances; (b) unifies identical instances into a frame; (c) adjusts the corresponding radio clocks; (d) unifies the white instances into the next frame.

on the transmitter’s address field (but these frames are not directly used for any higher-layer reconstruction, and any rare false matches will have little impact). Once Shaman has identified all the instances of a particular frame, it timestamps the frame using the mean of instance SUT timestamps.

Figure 4.5 illustrates the process of unification for two frame transmissions (dark and white circles). The figure shows the frames received by five radios R_i . Each column R_i corresponds to the queue of frames for that radio; in this example, three radios receive each transmission. Time flows down each column. Although a frame transmission is simultaneous, we represent skew among radios as circles at different time offsets. Figure 4.5a shows Shaman searching the radio queues within its search window defined by a time offset. It then compares frame contents, determines that they all are identical, and, as shown in Figure 4.5b, unifies the frames into a frame.

Clock adjustment

While the search window can accommodate slight variations in instance timestamps, it is inadequate to combat skew in the long term. Hence, we leverage the unification procedure to simultaneously resynchronize traces. When Shaman unifies a set of frame instances, the variances between the instance SUT timestamps and the frame’s SUT timestamp represent how much each clock now differs (again, it is critical that

we only use unique frames to drive this synchronization). The difference between this value and the timestamps on each instance represents a correction factor — positive or negative — that Shaman then uses to bring each of the associated traces back into synchronization. Figure 4.5b shows this correction as an adjustment of the time offsets for the frames in the queues of R_2 and R_3 , aligning the dark frames across radios to the offset of R_1 and effectively adjusting the offset of the white frame in the queue of R_2 .

A trade-off can be made between accuracy and the overhead of resynchronizing by placing a threshold on the minimum *group dispersion* — the difference between the earliest and latest SUT timestamps among all instances comprising a frame — before resynchronizing. Figure 4.5a illustrates the group dispersion for the first frame transmission as the difference in time offsets between the frame in the queues of radios R_2 and R_3 . In our implementation we set this threshold to $10\ \mu\text{s}$. (Note that this does not limit the synchronization accuracy to $10\ \mu\text{s}$.)

Managing skew and drift

If resynchronization happened frequently and uniformly across all traces, then it would be straightforward to maintain very tight synchronization bounds. However, there are frequently extended periods during which a particular radio may not observe any frames in common with others. During these times the synchronization of this radio’s observations is only guaranteed by the accuracy of its own local clock. Thus, the slope of its skew with respect to SUT will determine how quickly it will lose synchronization without re-adjustment.

In practice, we have found that, with large numbers of radios, it is not uncommon for two arbitrary radios to have large skews (over 10PPM) that may overhear frames sent from the wireless stations intermittently (every couple minutes). For example, if these two radios do not hear the same frames to synchronize their clocks over 1 minute, their clock would be off by $600\ \mu\text{s}$, which would exceed the synchronization search window. For such cases, we have to estimate the relative skews of these two radios. Scaled to the entire system, Shaman would need to track n^2 skews, where n

is the total number of radios. Updating all skews upon every resynchronization would then require $O(n^2)$ operations, imposing too much overhead in our system for real-time synchronization.

To solve this issue, for each radio Shaman maintains the skew to the SUT clock instead of skews to the other $n - 1$ radio clocks. Therefore Shaman only needs to update $O(n)$ radio clock skews for every new frame. The trade-off is less precise skews. However, we found this approach works very well in practice, and desynchronization is a rare event.

Figure 4.5c represents Shaman adjusting the skews of radios R_4 and R_5 by shifting the frames at the head of their queues. Shaman then repeats the unification process for the next set of frames in its search window, identifying the three white frames as identical. In Figure 4.5d, Shaman unifies them into a frame timestamped with the mean of the SUT timestamps of all frame instances. For this frame, however, the group dispersion is below the resynchronization threshold, and Shaman reduces overhead by skipping resynchronization for these frames.

Occasionally, it is possible to temporarily lose synchronization because the radio clock jitters or drifts. If two radios hear a 802.11 frame but the difference in local timestamps are outside the search window, Shaman will identify these two instances as two separate 802.11 frames. Shaman would not resynchronize these two radios, and hence the offsets and skews would not be adjusted. Consequently, Shaman would not be able to identify instances for later 802.11 transmissions observed by these two desynchronized radios. To detect such desynchronization cases, Shaman compares the current frame's sequence number with recent frames in the last 10 milliseconds. Shaman detects desynchronization if there are two frames using the same sequence number. Shaman will bind these two frames into one frame and resynchronize all radios that appear in the new frame. In our UCSD CSE network, Shaman typically experiences less than a hundred of such events per day.

Thus, Shaman can use almost all new “unique” frames for continual resynchronization. This approach presents several key advantages compared to approaches

that simply use reference beacons to synchronize [YYA04]. First, in large environments it is not possible to identify frames heard by *all* monitors and thus time synchronization must be transitive. Having more synchronization actions will almost always increase synchronization accuracy since the impact of clock skew is minimized. Second, since clients are mobile, their traffic creates a richer set of synchronization opportunities — touching pairs of radios that might never be directly synchronized otherwise. Finally, more clock samples allow for better management of skew and drift and therefore accuracy. In small-scale environments these factors may be minor. As the number of monitored radios increases, however, variability in skew, drift and workload conspire to raise the probability of a synchronization loss. This additional robustness becomes critical at a modest increase in complexity. Shaman’s synchronization code totals roughly 4,000 lines of C++.

Wall clock time calibration

It is important to calibrate the virtual SUT clock to wall clock time in our wired monitor for two reasons. First, this calibration synchronizes the time domains of the wireless and wired traces for our analysis. Second, it enables the diagnostic system that runs on our wired monitor to easily correlate times specified in user queries with timestamps in the trace. To support Shaman’s synchronization in the Shaman system, Shaman tracks the offset between the SUT clock and the wall clock of the wired monitor, and adds wall clock timestamps to the frames. Since our wired trace monitor also has a wireless radio that collects wireless traces, we slave the virtual SUT clock to the real-time clock by putting additional UTC timestamps in the traces of that particular wireless radio. Shaman then estimates the skews between the wall clock and jigsaw clock using linear square fits every 30 seconds.

Synchronization performance

Figure 4.6 illustrates the current accuracy of our algorithm using a 500 μ s search window. The graph shows the CDF of group dispersion values calculated for

Figure 4.6: CDF of group dispersion across all frames.

every frame processed from 193 radios over a 24-hour period. For 90% of all frames, the worst case time offset between any two radios is less than $10\ \mu\text{s}$, and 99% see a worst case offset under $20\ \mu\text{s}$. While the details of this graph are a function of individual clock characteristics, the network workload, and the number of clocks being kept synchronized, we believe it demonstrates that fine-grained broadcast synchronization is achievable in a building-scale environment.

Having precise short-term synchronization provides the necessary basis for our 802.11 timing analyses on the wireless traces. We do not need require such precision for the longer-term synchronization with the wall clock on the wired monitor. However, Shaman’s universal time is within 20 PPM skew to the wall clock UTC. This synchronization provides enough precision for us to diagnosis user reports and associate wired events and wireless events as discussed in the next chapter.

Shaman synchronization executes on our back-end server (a double AMD 2.2GHz Opteron) server. Shaman usually takes from 5 to 20 seconds to synchronize 1 minute of traces (10Mbps to 40Mbps) across all monitors. Thus, Shaman can perform

synchronization and unifications online and in real time.

4.3 Link-layer conversations reconstruction

Having constructed a single global view of each observed physical event, the next task is to reconstruct each link-layer conversation in its entirety. An 802.11 conversation includes the DATA and ACK frame deliveries and all local retransmissions. Each conversation represents a 802.11 transaction to delivery an Ethernet packet in or out of the wireless distribution network. Therefore, conversations are the basic unit for upper-layer protocol analysis.

The output of synchronization is a continuous stream of frames in a custom *jcap* format. A *jcap* trace has the format of a gzipped pcap file that combines a *jcap* header with every frame. The *jcap* header includes the transmission rate, preamble length, etc., in a 24-byte header for each frame. In principle, the 802.11 conversation reconstruction is straightforward since the frames in the *jcap* trace are time-ordered and each frame contains up to 200 bytes of payload that can be used to identify MAC addresses, IP addresses and TCP port numbers. In practice, however, missing data and vantage point ambiguities complicate this reconstruction process. Thus, Shaman must use inference to help reconstruct these higher-layer descriptions.

4.3.1 Station module

Most information required to reconstruct 802.11 conversations is directly available in the frame bodies, such as rate, frame length, and retransmission status. However, some characteristics of the sender and receiver must be inferred from the frame timings and AP beacons. The Shaman station module performs this task.

The station module tracks state related to a station's association with the network as a function of time. This state includes the preamble mode (short or long), slot time (short or long), power save mode (PSM), the use of 802.11g protection mode and RTS/CTS, etc. The station module maintains this state directly, by tracking the parame-

ters advertised in AP beacons and client scan probe frames, and indirectly, based on the timing of successive frame transmissions (e.g., sufficiently fast ACK responses preclude long preambles).

The station module reconstructs individual link-layer conversations using a *frame exchange constructor*. The constructor starts by identifying all *transmission attempts*. A transmission attempt usually consists of control frames, a data frame, and one ACK from the receiver (e.g., RTS-CTS-DATA-ACK is a common frame exchange pattern for 802.11g clients). The constructor then groups transmission attempts into complete *frame exchanges*. Since 802.11 implements ARQ for unicast frames, a frame exchange may involve multiple distinct transmission attempts. Since the wireless monitors can fail to capture some frames, the frame exchange constructor performs contextual inference to compensate for any omissions described in more detail below.

4.3.2 Link-layer inference

In reconstructing link-layer conversations, Shaman first identifies each *transmission attempt* from a sender (illustrated on the left side of Figure 4.7). For example, a CTS-to-self packet, a subsequent DATA frame and the trailing ACK response may all be part of the same attempt. To group these together automatically we first use the MAC address: DATA frames carry the address of the sender explicitly, CTS-to-self frames (used for 802.11g protection) do as well and ACK frames indicate the recipient's address. As well, we use the *Duration* field, carried in CTS and DATA frames, to deduce the future time in which an ACK, if sent, must have been received. This timing analysis is especially critical when frames are missing from the trace since otherwise we might risk assigning an ACK for a missing DATA frame to an earlier observed DATA frame. At the conclusion of this analysis stage, collections of one to three frames are associated into a single transmission attempt.

We then group transmission attempts into *frame exchanges* — complete sets of transmission attempts (including retransmissions) that end in a link-layer frame being successfully delivered or not. Since 802.11 implements ARQ for unicast frames, a

Figure 4.7: Finite state machines used to assemble individual frames into transmission attempts (left) and to compose transmission attempts into complete frame exchanges (right).

frame exchange may involve multiple transmission attempts. Normally it is sufficient to simply group nearby transmission attempts that share the same frame sequence number. However, when portions of transmission attempts are missing (*e.g.*, CTS and ACK, but not DATA), then we must deduce the presence or absence of this missing data based on the subsequent behavior of the sender and receiver.

We implement our inferences using a finite state machine capturing the visible aspects of the transmitter’s MAC state in addition to several heuristics (*e.g.*, that DATA is more likely lost than ACKs). We do not make inferences about frames for which we have no direct information (*i.e.*, sequence gaps greater than one) but our experience is that these situations occur rarely in our traces due to our dense deployment of monitors.

Broadcast and multicast frames (shown as *R1*) are never retransmitted, so transmission attempts and frame exchanges are identical. Frames without sequence numbers (*e.g.*, ACKs) are queued until more data becomes available to resolve their position. Unicast frames *with* sequence numbers are further classified based on the change in the 802.11 frame sequence number since the last transmission attempt from the same sender. Deltas of zero (*R2*) indicate retransmissions and both transmission attempts are coalesced into a single frame exchange. If the sequence number is incremented by 1 (*R3*), we can infer that a new frame exchange has begun, but it is ambiguous how precisely to assign the queued transmission attempts.

Thus, we use a number of heuristics based on empirical measurements (*e.g.*, almost all frame exchanges can complete within 500 ms, acknowledgments are less likely to be lost than data, the coded rate of a frame never increases in response to a loss, retransmissions usually have the retransmission bit set, etc.) to decide this issue. If the sequence increment is more than one (*R4*), we make no inferences, we flush the queue (these transmission attempts are unassigned) and assign the current transmission attempt to a new frame exchange. Overall, only 0.58% of the transmission attempts and 0.14% of the frame exchanges in our traces require some form of inference.

Finally, one of the most important questions we wish to infer is whether a particular frame exchange was successful, meaning the receiver has correctly received the data frame. Unfortunately, the vantage point of a passive monitor does not allow this situation to be determined unambiguously: if, after transmitting a DATA frame, we see an ACK, we can feel confident that the data was delivered. However, if we never see an ACK, it is ambiguous if the frame was lost or if we simply did not observe the ACK. However, we *can* disambiguate this situation by using transport-layer information.

4.3.3 Inference from TCP layer

We may process frame exchanges in which it is unclear if the frame was actually delivered (as described previously). We can use the transport-layer side effects of this frame exchange as an oracle to determine what truly happened. For example, a

data frame carrying a new TCP segment will cause subsequent TCP acknowledgments to “cover” its TCP sequence space. Thus, observing a covering TCP ACK *proves* that the link-layer frame containing the associated data was actually delivered. The second problem is that existing analyses assume that monitors are lossless (that is, they observe all packets that are delivered between endpoints). In the wireless context, even with many different monitors, sometimes a frame exchange is completed but not observed *at all* due to imperfect coverage describe in Section 4.1.4. However, if we observe a TCP acknowledgment that covers a TCP sequence hole, we can infer that the packet was correctly delivered. Thus, it is usually possible to infer the presence of any single packet omission at the TCP layer.

4.3.4 Cross reference of wired and wireless frames

Typically, a frame exchange represents a packet delivered either to or from the wired distribution network. Thus, successful frame exchanges usually also have a counterpart in the wired trace. Relating these two vantage points serves two important functions. First, the timestamp associated with the wired packet indicates unambiguously when the frame entered or left the wireless network. These events are critical for inferring detailed 802.11 behavior, which we discuss in detail in Section 5.4. Second, since the wired monitor does not drop frames it is critically useful in helping to identify any frames missing from the wireless trace. This is particularly useful for analyzing the TCP protocol, because such analyses can be quite fragile to missing packet data [MRWZ06].

The goal of the cross referencer is to match the packets in the wired trace (Ethernet II frames) to the frame exchanges on the wireless side with identical data content. It handles both one-to-one cases, where one wired packet corresponds to one wireless frame exchange (e.g., a unicast DATA packet), as well as one-to-many cases, such as when a single broadcast ARP request on the wired network induces broadcast frames at all APs.

The cross referencer adds a *matched frame exchange* structure into the trace,

linking the wireless trace representation of the packet (the frame exchange structure) with the wired representation of the packet (the captured Ethernet II frame) and combining the two input traces into one output trace. The transitive extent of a matched frame exchange can be considerable. Consider, for example, a DHCP request from a client to an AP. The client transmits the DHCP request to the AP as a unicast DATA frame destined to the AP. Because the request is also a network broadcast, the AP bridges it by broadcasting it on the AP wired distribution network. Each AP, including the bridging AP, will then transmit a broadcast frame for the DHCP request. The matched frame exchange for this scenario will therefore encompass the initial unicast frame exchange, the wired Ethernet frame for the bridged broadcast, and N frame exchanges for each of the broadcast frames from the N APs.

Shaman cross references by converting the wired Ethernet II frame into the expected 802.11 frame format, and performs a local search of the wireless trace to find the matching wireless frame. The search window is two seconds, which is based on the maximum of various wired-wireless forwarding latencies among AP queuing delay, 802.11 transmission delay, power-saving buffering delay, etc. For example, power-saving 802.11 clients wake up every DTIM interval, which is 102.4ms in our network. We have also found that our APs can queue packets up to 1.2 seconds. Therefore, we set a conservative 2 second maximum search window.

However, frame content matching may not be sufficient for some packets that have identical content and are sent within our two-second search window. One example is the wireless gateway, which typically sends three to five distinct ARP requests within 2 seconds. In this case, the cross referencer has to ensure the wireless and wired ARPs match are ordered appropriately.

Finally, the output of the cross reference module is the matched frame exchanges, which contain the measured, synchronized, and inferred wired and wireless information including timestamps on each network, frame bodies, etc. These matched frame exchanges are used to drive various analysis modules discussed in the next chapters.

4.4 Acknowledgement

Chapter 4, in part, is a reprint of the material as it appears in the SIGCOMM Conference, 2006, Cheng, Yu-Chung; Bellardo, John; Benko, Peter; Snoeren, Alex C.; Voelker, Geoffrey M.; Savage, Stefan. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, is a reprint of the material as it appears in the SIGCOMM Conference, 2007, Cheng, Yu-Chung; Afanasyev, Mikhail; Verkaik, Patrick; Benko, Peter; Chiang, Jennifer; Snoeren, Alex C.; Voelker, Geoffrey M.; Savage, Stefan. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Protocol Models and Analysis

In this chapter we develop a portfolio of comprehensive analyses on the building's wireless network captured by the monitor platform. We exploit the global perspective afforded by distributed monitors to answer operational questions (*e.g.*, what are the root causes of transient outages and performance degradations) and research questions (*e.g.*, how to better optimize wireless protocols and their interactions with other protocol layers).

We start by summarizing high-level characteristics of the trace of a typical weekday taken in our monitoring platform. Then we walk through the possible problems an 802.11 frame could encounter in the wireless distribution network to motivate our analyses. As depicted in Figure 3.1, our analysis portfolio consists of a broad-band interference analyzer, co-channel interference analyzer, DHCP, ARP, 802.11 scan/association analyzer, and MAC delay and TCP performance analyzers. Not all the analyzers are presented in this dissertation because they are designed and implemented by collaborators. We refer readers interested in the analyzers for PHY broadband interference, DHCP/ARP, 802.11 association to [CAV⁺07a, CAV⁺07b]. The chapter details the remaining analyzers.

Table 5.1: Summary of trace characteristics.

Start	1/24/06 @ 00:00
Duration	24 hours
Radio Monitors	156
Total APs	107
Our APs	39
Other APs	68
Our Clients	1,026
Total Events	2,700 M
Physical Errors	338 M (13%)
CRC Errors	956 M (35%)
Valid Frames	1,410 M (52%)
frames	530 M
frame instances	1,580 M
Instances/frame	2.97

5.1 Trace summary

We start by summarizing the high-level characteristics of our trace and then show network activity over time. Table Table 5.1 presents the characteristics of the trace we use for our analyses. The trace captures traffic for the entire day of Tuesday, January 24, 2006, a typical workday in our building. Just as APs within buildings are not isolated, buildings themselves are not isolated: we observe traffic associated with more than twice as many APs in surrounding buildings than in this one. For the subsequent analyses, though, we focus only on the traffic generated by clients associated with our APs; our monitors cannot capture traffic from external APs with good coverage due to their remote location. We see 1,026 unique client MAC addresses associated with our APs during the day.

Throughout the day the monitors observe over 2.7 billion instances. Over 47% of these instances are physical or CRC errors. This high percentage is not surprising given transmissions observed by distant monitors just beyond reception range, the presence of both co-channel interference (hidden terminals) and broadband interference (microwave ovens), etc. Shaman unifies 1.58 billion instances (valid frames and a subset

(a) Active Clients and APs

(b) Traffic Breakdown

Figure 5.1: Time series of network activity throughout the day in one-minute intervals.

of associated error frames) into 530 million frames, for an average of 2.97 instances per frame. In other words, on average the monitoring platform makes three observations of every observed transmission of a valid frame in the network.

Figure 5.1 shows network activity as a time series throughout the day at the granularity of one minute. Figure 5.1(a) shows the number of active clients and APs per one-minute time slot as a stacked bar graph. We define an active client as one that is communicating with an AP or is actively establishing an association. An active AP is one communicating with an active client (an AP only sending out beacons, for example, would not be active). Activity exhibits an expected diurnal pattern. Most clients are active from late morning (10am) until late afternoon (5pm), with many clients active in the early morning and well into the night. The number of active APs grows as more clients become active throughout the building. The clients active overnight are likely wireless devices without user activity, such as wireless laptops left running with applications that produce background traffic.

Figure 5.1(b) shows the amount of traffic per one-minute time slot as a stacked bar graph of four traffic categories. “Data” counts both unicast and broadcast data frames, and “Management” counts various management and control traffic (RTS/CTS, ACKs, association, *etc.*). Although the number of active clients is relatively smooth

over time, the traffic generated by those clients is much more bursty. Many of the bursts start on an hour or half-hour time boundary, likely indicating laptop usage during meetings and talks in the building. Since most management and control traffic relates to data traffic, it closely tracks the amount of data traffic.

We also separate out two explicit categories of management traffic because of their high prevalence: “Beacon” shows the amount of periodic AP beacon traffic, and “ARP” shows the amount of ARP broadcast ARP traffic. Because APs broadcast beacon traffic independent of activity, beacon traffic is constant throughout the day. ARP traffic is more interesting. In addition to legitimate use, outside scans and worms generate ARP traffic as they probe unallocated IP address space. However, it appears that the largest source of ARP is due to an 802.11 management server from Vernier that uses regular ARPs to track the liveness and network location of registered clients. However, the important aspect of ARP traffic is that it is broadcast. Because 802.11 APs are designed to act as transparent bridges all ARP “who-has” broadcasts from the wired network are also broadcast on the wireless channel. Since broadcast frames are always encoded at the lowest rate they make highly inefficient use of the medium. Indeed, if we examine our trace strictly from an *air time* perspective, broadcast traffic (primarily ARP and Beacons) regularly consumes 10% of the channel as seen by any given monitor. Finally, because they are delivered to all APs at the same time, they are broadcast on all APs on all channels at roughly the same time as well — likely interfering with themselves in the process.

Indeed, all network-layer broadcast traffic has this side effect, including client DHCP requests and application broadcasts.¹ Moreover, aspects of this traffic scale with the size of the network or the size of the user population while the capacity of the channel remains constant. Thus, we argue that applications should use multicast instead of broadcast on 802.11 networks and 802.11 APs should be modified to perform selective filtering of non-unicast traffic. Finally, to eliminate the implicit synchronization caused

¹One particularly egregious example (almost 100,000 frames in our trace) is the Mac version of the MS Office suite. As part of an anti-piracy mechanism the software regularly broadcasts its license information to UDP port 2222.

by wired broadcasts, APs should add random jitter to the transmission time for broadcasts frames received from the wired network.

5.2 The troubled life of a packet

There are numerous sources of disruption or performance degradation in an 802.11 network. To illustrate these challenges and motivate the need for our analyses, we provide a quick primer on several potential sources of delay and packet loss.

5.2.1 Physical layer

The physical layer presents the first obstacle for an 802.11 frame hoping to be delivered. Sharing the unlicensed 2.4GHz ISM band are a wide range of non-802.11 devices, ranging from cordless phones to microwave ovens. An 802.11 packet in flight may be corrupted by broadband interference from such devices or it may simply be overpowered at the receiver. Alternatively, the sender may detect the presence of RF energy on the channel and defer transmission — incurring delays until the interfering source ceases.

For example, Figure 5.2 illustrates the interference caused by a microwave oven. The figure depicts the reception of physical error frames over time. The characteristic pattern (the white gap) results from the wave doubler circuit used in consumer microwave ovens to convert A/C line power into microwave energy. Roughly speaking, a U.S. oven will generate swept broadband interference for 8 ms (half of the 60-Hz cycle) followed by a similar period of quiescence. In all cases, in-range 802.11 radios will defer transmission until the medium is idle, building queues and adding delay in the process. Frames in flight when the oven is turned on may be corrupted, depending on the receiver power of the microwave signal.

Such physical layer interactions are not restricted to non-802.11 devices. The 2.4GHz ISM band combined with the nominal 22-MHz channel bandwidth used by an 802.11 transmitter can easily overlap neighboring transmitters on different channels.

Figure 5.2: Physical error frame pattern during microwave oven use. The y-axis depicts $(\text{time} \% 16)$ ms, showing the offset of physical error packets within the 16 ms microwave period. The x-axis shows two minutes of microwave use.

Indeed, while conventional wisdom holds that 802.11 has three orthogonal channels in the United States, this statement is not always true in practice. We have routinely observed adjacent channel interactions and have even witnessed many successful packet receptions between radios in which the transmitting and receiving radios were separated by as much as 50 MHz (i.e., channel 1 to channel 11). In addition to neighboring channel interference, 802.11 also suffers from the *capture effect* [KVSA04], which means a radio often decodes the frame with higher signal strength when two packets collide at a receiver.

5.2.2 Link layer

The 802.11 link-layer presents another potential performance land mine for user packets. In particular each 802.11 access point manages two critical functions: media access and bindings between individual stations (clients) and APs. Each of these functions can induce additional and, at times, unnecessary delays. We consider each in turn.

Transmission delays. Sources of link-layer transmission delay include queuing at the AP prior to wireless transmission, protocol delays such as mandatory backoff on transmission, exponential backoff on loss, packet transmission time (a function of the encoded frame rate and the packet size), and contention in the network when users and APs overlap and share a contention domain (or due to interference as mentioned above). A single packet may be delayed by all of these factors and, due to retransmission, it may be impacted multiple times. Moreover, it is common for 802.11 drivers to encode data at a lower rate after a loss, even though this practice may have unintended negative effects such as increasing channel utilization.

For example, consider a packet received by an AP at time t . It may be delayed in a queue waiting for previous packets to be transmitted (each experiencing their own media access delays and retransmission overheads). When it reaches the head of the queue the AP must perform a mandatory backoff, waiting between 0 and 15 slot times (a normal 802.11b slot is $20\ \mu\text{s}$, although 802.11g permits the use of a “short” $9\text{-}\mu\text{s}$

slot time under certain circumstances). After the backoff it must sample the channel for the duration of a “DIFS” interval ($50 \mu\text{s}$) before sending. If the AP detects a busy channel, it will perform yet another backoff before commencing the transmission. Finally, the packet is transmitted with a delay largely determined by the sender’s choice of rate. However, if the sender does not receive an acknowledgment from the receiver, the sender performs another backoff before each retransmission. Of course, this explanation is over-simplified and any real analysis must also deal with delays from interacting protocol features like power management and vendor irregularities (e.g., some vendors allow certain packets to be prioritized in between retransmissions of a frame exchange). Unfortunately, most of the delay components at this level cannot be observed directly since they depend on the internal state of an AP, which is not exposed via any protocol feature.

Management delays. Another important source of overhead in wireless networks broadly falls into the category of wireless management. 802.11 clients and APs are in a constant dance trying to determine the best pairing. To address issues of mobility, clients continually scan their environment looking for a better partner. APs respond to these scans, and additionally broadcast beacons to nearby clients. If a client switches APs, another set of exchanges takes place that authenticates the client to the network and binds the client and the AP (a process called association).

Additionally, APs must deal with significant heterogeneity in their client base, which includes distinct capabilities and configurations. Consequently, a negotiation takes place between clients and APs about which features are needed — 802.11b vs. 802.11g transmission, power savings, etc. Unintuitively, the choice of a single notebook computer to associate with an AP can transform that AP’s behavior as it tries to accommodate the lowest common denominator among its clients. For example, in Section 5.5 work we reported that the presence of a single 802.11b client — even one that is not transmitting — will often force an AP into 802.11g “protection” mode, thereby degrading service for all 802.11g users. [CBB⁺06]

5.2.3 Infrastructure support

APs are fundamentally bridge devices. To obtain Internet connectivity a client must in turn acquire an IP address — typically via DHCP — and the MAC addresses of next-hops to destinations — typically via ARP. These protocols exhibit complex dynamics in themselves, and their failure may isolate a station for some time. Their use with 802.11 exacerbates their complexity since they are used in specialized ways, frequently tied together with VLANs using proprietary mobility management software that authenticates stations via a single sign-on interface and allows IP addresses to remain constant as a client roams between APs. There is no standard for implementing this functionality and, unsurprisingly, failure modes are not well understood.

5.2.4 Transport layer

Finally, any underlying delays or losses are ultimately delivered to the transport layer, usually TCP, which may amplify their effects believing these behaviors to be indicative of congestion.

While this complex set of processes frequently works surprisingly well, when it does not it can fail spectacularly and expose users to significant response time delays. It is the goal of last part of this dissertation to systematize the analysis of these issues to better understand the source of such transient problems.

5.3 Modeling 802.11 frame loss

In this section, we analyze the extent of transmission losses experienced by nodes in our trace. Losses can be caused by many factors including signal attenuation, hidden-terminal interference from 802.11 devices or non-802.11 devices, etc. The signal attenuation (weak signal) is usually easier to detect and resolve from the APs or the clients perspective. Indeed, most vendors implement solutions to enhance the signal strength by either hardware (better antenna) or software (reduce transmission rate). However, hidden-terminal interference is much harder to diagnose from the network

perspective. We will describe how Shaman can analyze the losses caused by hidden-terminal of other 802.11 devices on a per sender-receiver pair basis next. The non-802.11 broadband interference detection is in [Chi06]

5.3.1 Hidden-terminal interference

Since the platform monitors orthogonal channels, adjacent-channel interference is rare and co-channel interference from hidden terminals is likely the dominate cause of interference. As a result, the distributed monitoring platform provides the key ability to observe co-channel interference. By providing a global perspective on the network, we can simultaneously detect a transmission from a sender to a receiver, hypothesize that the transmission was lost, and detect that a third node was transmitting at the same time as the sender. With only a single vantage point, it would be very difficult to detect and correlate such simultaneous transmissions.

We define an interference event as a unicast transmission from a sender s to a receiver r in which one (or more) interferers i simultaneously transmit and cause the transmission from s to r to fail. Based upon events in the trace, our goal is to estimate what fraction of these simultaneous transmissions causes a loss due to interference. Note that packet transmissions are distinct from frame exchanges; a successful frame exchange might experience multiple transmission losses and recover using link-level retransmissions.

We measure simultaneous transmissions when the trace contains more than one transmission overlapping in time during which s transmits a packet to r . We infer that the transmission from s failed to reach r when we do not observe an ACK from r . At this point, though, when a loss happens we cannot say for certain that a particular simultaneous transmission was the true cause of the loss. It may be the case that a node in a remote part of the building just happened to have transmitted at the same time that a transmission from s to r was lost; *i.e.*, the loss may have been caused by any number of reasons entirely unrelated to the remote node's transmission.

We can, however, infer when losses are likely due to simultaneous transmis-

sions. In particular, we can infer the conditional probability P_i of a simultaneous transmission causing interference given that there is a simultaneous transmission from s to r . We can infer P_i based upon the losses between s to r when simultaneous transmissions both do and do not occur. Informally, if we assume that the background loss rate is constant regardless of the number of transmissions, we can attribute the losses between s and r during simultaneous transmissions accordingly: If s and r experience few losses in the absence of simultaneous transmission, the more likely the losses they experience during simultaneous transmission are due to interference.

More formally, let I be the event that interference causes a lost transmission from s to r , and L be the event that the transmission from s to r was a background loss due to some other cause (*e.g.*, range, obstacles). Let S be the event that there is a simultaneous transmission from at least one other device i when s transmits to r . Note that I and L are independent events. For the case where no multiple simultaneous transmissions occur, $P[I|\neg S]$ is obviously 0. Unfortunately, when there are multiple transmissions we cannot empirically distinguish between I , L , or $(I \cup L)$ upon observing a loss. We can, however, calculate the probability of interference when there is more than one simultaneous transmission as follows:

$$P_i = P[I|S] = P[(I \cup L)|S] - P[L|S] + P[(I \cap L)|S].$$

We can calculate this conditional probability based upon events measured in the trace. For a given (s, r) pair, let n be the number of transmissions from s to r , $n_0 \leq n$ be the number of transmissions from s to r *without* a simultaneous transmission from another node, and n_0^l be the number of n_0 transmissions lost. Likewise, let n_x be the number of transmissions from s to r *with* a simultaneous transmission, and n_x^l be the number of n_x transmissions lost.

Then we can measure $P[(I \cup L)|S]$ empirically as n_x^l/n_x . Observing that L is independent of S , the case of simultaneous transmissions, we have $P[L|S] = P[L|\neg S] = n_0^l/n_0$ and $P[(I \cap L)|S] = P[I|S] \cdot P[L]$. A bit of algebra then reveals:

$$P_i = P[I|S] = [(n_x^l/n_x) - (n_0^l/n_0)] / (1 - n_0^l/n_0).$$

Figure 5.3: Interference loss rate X across (s, r) pairs.

Given P_i , we can then estimate the expected number of losses during simultaneous transmissions between an (s, r) pair that are due to interference. Examining all transmissions between all sending and receiving pairs, we can estimate the extent to which interference occurs in our network.

We restrict our analysis to the 536 (s, r) pairs that exchange at least 100 packets to provide confidence in our statistical estimates. These (s, r) pairs comprise 82% of all (s, r) pairs in the trace. All such pairs experience losses with at least one simultaneous transmission. Normalizing these losses according to the background loss rate of each pair according to the above formula, we estimate that 88% of these (s, r) pairs experience loss due to interference from another node. Whose transmissions are being interfered with? Of those (s, r) pairs experiencing interference, the sender s is split roughly equally between APs (56%) and clients (44%).

Does interference have a significant impact on the overall transmissions from senders to receivers? Again, note that lost transmissions may increase frame exchange times due to retransmissions, but not necessarily result in a failed frame exchange. To

answer this question, Figure 5.3 shows the *interference loss rate* as a CDF across all (s, r) pairs. We define interference loss rate X as the fraction of all transmissions (*i.e.*, regardless of whether there was a simultaneous transmission or not) from s to r that were lost due to interference; alternatively, it is the probability that a transmission from s to r is lost due to interference:

$$X = P_i * (n_x/n)$$

As a baseline, the average background transmission loss rate is 0.12. In comparison, the results in Figure 5.3 show that many (s, r) pairs experience minor interference: 50% of (s, r) pairs experience an interference loss rate of 0.025 (a 2.5% probability of a transmission lost due to interference), or less. Yet a noticeable fraction of (s, r) pairs suffers considerably from interference: 10% of pairs experience an interference loss rate of at least 0.1, and 5% at least 0.2. A few (s, r) pairs experienced terrible interference with an interference loss rate higher than 0.5. Note that it is possible for P_i to be negative; in these cases (11% of pairs), we truncate X to 0.

5.4 Modeling 802.11 frame delay

In this section we describe a media access model for measuring and inferring the critical path delays of a monitored frame transmission.

The model consists of a representation of the wired distribution network, queuing behavior in the AP, and frame transmission using the 802.11 MAC protocol. The goal of the model is to determine the various delays an actual monitored frame encounters as it traverses the various stages of the wireless network path. At a high level, our approach first determines a series of timestamps for a frame as it traverses this path and is finally transmitted by the AP. From these timestamps we can compute the delays experienced by the packet. Table 5.2 summarizes the definitions of the timestamps and delays in our model, and Figure 5.4 illustrates where in the network path they occur.

Our model uses measurements of the frame both on the wired network and the wireless network to determine some of the timestamps. The challenge, however, lies

in inferring the remaining timestamps and, hence, delays. The inference techniques we develop, along with the representations of AP queuing and the transmission behavior necessary to perform the inferences, represent a key contribution of this dissertation.

In the following sections we describe in detail our model components and how we measure and infer these timestamps and delays. We then show how the critical path delays determined by the model can provide valuable, detailed insight into the media access behavior of wireless users. Finally, we show how we can use the model to diagnose problems with TCP throughput.

5.4.1 Critical path timestamps

To start our analysis, we first measure the timestamp of each packet as it leaves the wired gateway router on the way to a wireless access point — a time we define as t_w . We capture this information using a SPAN port configured to forward a copy of each packet as it leaves the building’s main distribution router. These copies are directed to a dedicated tracing server where they are timestamped (we assume that this propagation delay is constant).

To calculate additional timestamps we must combine observations of the packet on the wired network together with observations of the packet on the wireless network. To match packets across wireless and wired traces, we compare normalized packet contents (adjusting for 802.11 vs Ethernet II frame formats) over a one second window; one second reflects the empirical maximum wireless forwarding delay of a wired packet in our network. Most matches are one-to-one, meaning one wired packet corresponds to one wireless packet, but there are cases of one-to-many matches. For instance, broadcast frames such as ARP requests can match multiple wireless frames because each AP will forward the ARP request to the wireless network. Occasionally, a packet is also dropped due to AP queue overflows — typically when clients perform bulk downloads — which we detect based on frame sequence numbers. Overall we match 99.95% of the wired frames in our trace.

The next step is to determine when the AP has received the frame from its

Figure 5.4: Representation of wired network, queuing and transmission behavior in the AP. The arrows indicate where network we measure/infer timestamps as frames traverse the network.

wired interface. Since we do not have taps on the APs or control the AP software, we cannot directly measure this time, t_i , and instead must infer it. t_i is a function of the AP's Ethernet I/O delay and the propagation delay between the gateway router and the AP. For each AP, we estimate t_i by first measuring the distribution of the interval $(t_s - t_w)$, the difference between the wireless transmission time and the wired timestamp of the packet. The minimum value of this distribution, minus DIFS, is the sum of wired network delay and AP input processing delay for the minimum packet size.

From here, we determine the transmit queue timestamps of the packet inside the AP, both when the packet enters the transmit queue (t_q) and when it reaches the head of the queue (t_h). We model the AP as having three FIFO packet queues, the transmit ready queue and two waiting queues based on the 802.11 standard. If the packet is broadcast or multicast, the AP schedules it onto the broadcast queue; the AP flushes this queue into the transmit queue after the next beacon transmission. If the packet is destined to a power-saving client, the AP buffers it on a power-save queue. The AP

Table 5.2: Summary of timestamps and delays determined by the media access model.

Timestamp Definitions	
t_w	Frame leaves gateway
t_i	AP receives frame from wired interface
t_q	Frame enters radio transmit queue
t_h	Frame reaches head of transmit queue
t_s	First bit of the frame transmitted
t_e	End of last ACK or estimated ACK timeout
Delay Definitions	
d_{ps}	$t_q - t_i$: AP power-save/broadcast buffering delay
d_q	$t_h - t_q$: AP (transmission) queuing delay
d_{mac}	$t_e - t_h$: MAC delay
d_{t0}	$t_s - t_h$: Access delay of first transmission attempt

flushes the appropriate packets from the power-save queue into the transmit queue after the client wakes up (by receiving a PSM-reset data or management frame, or a PsPoll frame from the client). Otherwise, the AP places the packet directly on the transmit queue.

It is critical to model the queuing behavior precisely to estimate further wireless delays. For example, if we did not model packets sent to clients in power-save mode correctly, they would appear to be delayed at the AP for tens of milliseconds. We determine whether clients are in power-save mode when packets for them arrive at the AP by tracking either the PSM bit of client frames in the wireless trace, or when beacons indicate that the AP has buffered packets for clients (TIM). Further, the 802.11 standard dictates that an AP should deliver broadcast frames at beacon intervals if power-saving clients exist because these clients only wake up at those times.

Based on the frame destination and client power status, we tag each frame with the appropriate queue type. Subsequently, we estimate the time when the AP places the frame on the transmission queue, t_q . For a broadcast/multicast frame, t_q is the time of the latest beacon prior to the frame's transmission. For frames destined to power-saving clients, t_q is the time the client notifies the AP that it has woken up by sending a frame with the PSM bit off such as a PsPoll control frame. For the remainder of the frames, t_q

$= t_i$ because the AP schedules them on the transmission queue immediately after it has received them from the wired interface.

Next, we infer the time when the packet reaches the head of the queue, t_h , and the AP is ready to transmit it using 802.11 DCF. We determine t_h under three conditions based upon the end time of the previous frame exchange, t_{pe} . First, if the AP places the frame on the transmit queue before the previous transmission completes, then the frame experiences head-of-line blocking. We conclude the frame reaches the head of the queue after the previous frame exchange finishes ($t_h = t_{pe}$), and we label this frame as “head-of-line blocked.” According to the 802.11 standard, a sender must perform a mandatory backoff at the end of each frame exchange to provide fair channel access. We cannot directly measure this random backoff window but we know the maximum of this window from the standard. Therefore, if the frame enters the queue beyond the maximum mandatory backoff window after t_{pe} , the frame must find the transmit queue empty and the AP can transmit immediately. Hence, $t_h = t_q$, and the packet is labeled as “not head-of-line blocked.” Finally, if the AP places the frame on the transmit queue during the maximum mandatory backoff window of the previous attempt, the frame may or may not experience head-of-line blocking by the previous frames. Since this backoff window is very small (300 μ s in 802.11g), less than 1% of the frames fall into this category. We assume the transmit queue was empty at t_q and the frame does not encounter head of line blocking. Thus $t_h = t_q$ as well.

We determine the starting and ending transmission times of the frame exchange, t_s and t_e , directly from the synchronized trace. The start time t_s is the start of the first transmission attempt, including the control overhead of RTS/CTS and CTS-to-self. The end time t_e is the end of the frame exchange: the end of the ACK of the last transmission attempt, including all retransmissions and contention. For unacked broadcast frames, t_e is the scheduled end time of the transmission (NAV end). Consequently, for unacked broadcast frames t_e is the end time of the data frame plus 60 μ s.

Frames internally generated in the AP represent a special case because we cannot observe when the AP generates them. For example, we do not know when the

AP has scheduled a scan response because no corresponding packet appears in the wired trace. Fortunately, these frames are typically management responses to client requests, such as scan responses and association/authentication responses. We assume that the AP generates these responses and places them on the transmit queue (t_q) immediately after it receives the requests.

5.4.2 Critical path delays

We calculate the critical path delays as intervals between timestamps. In particular, the buffering delay for power-saving clients and broadcast frames is $d_{ps} = t_q - t_i$, the time from when the frame reaches the AP and when the AP places the frame on the transmit queue. We label this “power-saving delay” because broadcast frames are buffered for power-saving clients who periodically wake up at beacon intervals. The AP transmission queuing delay is $d_q = t_h - t_q$, the time between when the AP places the frame on the queue and the time when it reaches the head of the queue (i.e., the AP is ready to transmit it). After the frame reaches the head of the transmit queue, d_{mac} is the time the AP takes to perform a frame exchange to the receiver including clear channel assessments, PHY (re-)transmissions, and any exponential backoffs. Thus, $d_{ps} + d_q + d_{mac}$ is the total time the packet spends in the wireless distribution network.

We further categorize the queuing delay d_q into three components: delay caused by background management frames such as beacons, scan responses, etc. (d_{qb}), unicast frames to the same client (d_{qs}), and unicast frames to other clients (d_{qo}); $d_q = d_{qb} + d_{qs} + d_{qo}$. These values are calculated by modeling the contents of the AP queue, characterizing frames queued earlier and summing their media access delays (d_{mac}).

5.4.3 Validating the model

To validate our AP model, ideally we would instrument an AP and compare our inferred timings and the actual ones for every frame transmitted. Unfortunately, we do not have access to commercial APs or open-source 802.11 drivers that export queuing or channel-probe delay timestamps on a per-frame basis. However, we can examine the

```

N = CWmin
1. Wait DIFS until channel becomes idle.
2. If channel is not busy, go to step 4.
3. Perform a regular backoff
   bo = rand[0, N]
   While bo > 0
       probe channel for 20us
       if busy wait DIFS until idle.
       --bo
4. Send the frame; if no ACK is received,
   double N and retry from step 1.
5. N = CWmin, perform a mandatory backoff
   as in step 3.

```

Figure 5.5: Simplified 802.11 DCF operation for unicast.

delays inferred by our model and determine whether those delays are consistent with delays expected from the known operation of the 802.11 MAC protocol.

First we examine the distribution of the access delay of the first transmission attempt, $d_{t0} = t_s - t_h$, from a TCP flow from our trace. Figure 5.6 shows the cumulative distribution of d_{t0} in microseconds for one hour of frame exchanges that are “head-of-line blocked” from an Avaya AP to a client using 802.11g to perform a bulk TCP download. Most of the traffic from the AP is destined to that client during that hour. We focus on the first transmission attempt of head-of-line blocked frames (typical for bulk downloads) because of the predictable delay distributions that should result from the 802.11 protocol.

To explain the distribution, we first summarize the 802.11 transmission process in Figure 5.5. This code segment is a simplified version of the unicast DCF operation in the 802.11 standard [Com99]. For frames sent in succession, the AP first waits a

mandatory backoff delay. The mandatory backoff delay is $bo \cdot 20\mu s$, where the Avaya AP randomly chooses the integer slot bo between 0–15 for 802.11g. After the mandatory backoff, the AP will start a regular DCF operation. First it listens on the channel for the DIFS interval ($50\mu s$). If the channel is idle, the AP transmits the frame immediately. In this case, d_{t0} is the mandatory backoff delay plus the DIFS delay. During the backoff, the AP defers decrementing bo until the channel becomes idle for DIFS. Therefore the backoff delay depends on a combination of bo and the channel contention the AP experienced.

The distribution of access delays shown in Figure 5.6 reflects the various components that comprise the overall access delay d_{t0} . Every frame must wait at least a DIFS interval during the transmit process; hence, the distribution starts at a delay of $50\mu s$ marked by the first vertical line. The “steps” immediately following correspond to the mandatory backoff delay that does *not* experience any contention. The frames have a DIFS delay plus the mandatory backoff delay, a random multiple of $20\mu s$ slots from 0–15; each step in the graph corresponds to one of the slots. The second vertical line ($X = 50 + 15 * 20$) marks the end of this category of frames (about 60% of the frames transmitted).

The next group of frames (through $847\mu s$) are frames experiencing contention during the mandatory backoffs. The contention they experienced is mostly due to TCP-ACK packets from the client to the AP. The PHY transmission time of these packets is $447\mu s$. As a result, the backoff incurs an additional $447 + \text{DIFS} = 497\mu s$ contention delay — hence the second set of “stairs” that starts at DIFS (from step 1) + $497 = 547\mu s$ and ends at $547 + 15 * 20 = 847\mu s$. The second set of stairs is not as pronounced because the sender may experience different lengths of contention delays. The remaining 10% of transmitted frames with the largest delays are frames that experienced longer contention delays or performed a regular backoff in step 3 of Figure 5.5.

Notice in the first set of stairs that the later steps tend to be shorter than earlier steps, but the second set of stairs show the opposite pattern. This behavior is because, having chosen a larger bo value, the sender is more likely to lose the contention and

Figure 5.6: Access delay (d_{t0}) distribution of one hour of head-of-line blocked frame exchanges from an Avaya AP-8 AP to a client doing a bulk TCP download.

to have to wait for the winner to finish its transmission. The probability of getting interrupted during mandatory backoff is about 40% (the percentage of frame exchanges experiencing congestion in the analyzed flow), which roughly corresponds to the delay ACK policy in TCP (send ACK on every other TCP-DATA).

Next, we perform a similar analysis using an AP from a different vendor to show that our approach is not tied to the implementation of a particular vendor. Since we could not replace an Avaya AP in our production network with an AP from another vendor, we instead performed a controlled experiment using a Cisco Aironet 350 AP. We downloaded a large file using scp to a client connected via the Cisco AP using 802.11b/g, and we plot the d_{t0} delay distribution as the “all” line in Figure 5.7.

At first glance the distribution looks dramatically different from the Avaya’s distribution in Figure 5.6. But, in fact, it reflects the same 802.11 sender transmission process, albeit using different parameters. The parameters differ from Avaya because the Cisco AP only used 802.11b, so its minimum contention window, CW_{min} , is 31

Figure 5.7: Access delay (d_{t0}) distribution of head-of-line blocked frame exchanges from a Cisco Aironet 350 AP to a client doing a bulk scp download.

instead of the 15 used by 802.11g. As above, the distribution is a mix of two kinds of frame exchanges. The first kind are frame exchanges that experience contention during the mandatory backoff. If we detect that the AP has acknowledged some frames (mainly TCP-ACKs from the client) during the mandatory backoff in step 5, we label the frame exchanges as having contention and plot the distribution as the “contention” line in Figure 5.7. The line forms a set of stairs starting from around $400 \mu s$. This offset is exactly DIFS plus the pause during backoff to wait for a TCP-ACK transmission. Otherwise, we plot the remaining frame exchange delays in the “no contention” line, which forms another set of 31 steps starting at DIFS. If we aggregate these two distributions, it forms the “all” line analogous to the curve shown in Figure 5.6.

We have also performed a similar experiment with the Avaya AP-8 where we change the slot time to the short slot time ($9 \mu s$) instead of the regular slot time ($20 \mu s$). The distribution changes (the width of a stair) accordingly.

In summary, even though we must infer the timing of some of the events that

Figure 5.8: Critical path delays, goodput, and losses over time for frame exchanges from an AP to a client doing a bulk TCP download.

determine critical path delays, our experience has been that our media access model is consistent with 802.11 operation even for very fine-grained phenomena. Furthermore, we can apply our model to different vendor APs, parameterized accordingly. Fortunately, these parameters are straightforward to obtain. The model requires 802.11 parameters like minimum contention window, maximum contention window, and slot time, all of which can be found in the AP manual or configuration GUI to correctly parameterize the model for a deployment.

5.4.4 Applying the model

The media access model makes it possible to measure the critical path delays for every packet sent from APs to the client. As an example, we focus on a particular AP in the building where three clients (X_b , Y_b , Z_g) are using TCP to download different files from the same Internet server, and the downloads overlap in time. The clients compete with each other for both AP resources and air time. Two clients use 802.11b (X_b , Y_b) and the third uses 802.11g (Z_g).

We apply the media access model to Y_b 's TCP flow to measure the critical path delays for each of the packets sent from the AP to the client. Figure 5.8 shows the delay breakdown for this client's packets over four minutes. Each spike in the graph corresponds to the combined queuing and wireless transmission delays for transmitting one frame. The MAC delay, d_{mac} , is quite small (even with contention among three clients) and are shown at the top tip of each spike. We break down the queuing delay into

three components: “other” is the delay d_{qo} waiting for frames to other clients to leave the queue; “self” is the delay d_{qs} waiting for frames to this client; and “background” is the delay d_{qb} waiting for background management frames (beacons, scans, etc.). Overlaid across the spikes is the TCP goodput achieved by the client. Above the spikes we show points in time where a frame was lost during wireless transmission (triangles) and on the Internet (diamonds).

This detailed breakdown shows a number of interesting interactions and behavior. First, queuing delay in the AP is the dominant delay on the wireless path to the client. These delays are orders of magnitude larger than the wireless transmission delay. Second, roughly half of the time client Y_b ’s frames were queued for its own frames, and the other half was caused by delays encountered by frames for the other two clients. Examining the frame delays of the other clients, most of those other frames were for client X_b and the minority were for Z_g . Third, Y_b experiences occasional wireless loss, but wireless loss does not have a substantial impact on achieved goodput. Fourth, Y_b experiences a burst of Internet loss at 14:39:38, substantially impacting goodput. The AP queue drains as Y_b times out and recovers. Finally, Y_b ’s download goes through a phase change just after 14:40:00. The other clients finish downloading (the frames in the AP queue are for Y_b) and Y_b no longer has to share the channel. AP queue occupancies drop and goodput increases substantially beyond the level when it was contending with other clients.

5.5 802.11b/g compatibility overhead

Next we analyze the use of 802.11g protection mode in the network. We find that the protection policy by our APs is overly conservative, potentially reducing performance for 802.11g clients. We then take advantage of the global perspective provided by the distributed monitoring platform to estimate the number of 802.11g clients that would benefit from using a more practical 802.11g protection mode policy.

During busy periods, we found a high rate of CTS control frames in the trace.

Investigating further, we determined that these are primarily CTS-to-self frames used for 802.11g protection. Since protection mode increases delay and reduces throughput for 802.11g clients, APs should only use protection mode when any active 802.11b clients are in range. The APs in the network implement this protection policy, but with an overly conservative timeout. An AP will not turn off protection until an hour has passed without sensing an 802.11b client in range.

In this analysis, our goal is to identify which APs in the trace are using protection mode that unnecessarily impacts 802.11g clients; we refer to these APs as *overprotective* APs. We can identify the set of APs using protection mode based upon CTS-to-self client transmissions to those APs. Then, using the global perspective of the unified trace, for each AP using protection mode over time we can infer whether any 802.11b clients are in range of that AP after a more practical timeout of one minute. If no 802.11b clients are in range, then the AP is overprotective. Using observed probe responses, we infer whether any 802.11b clients are in range of an AP using protection mode. APs send these frames after they receive a corresponding probe request from a client. Our monitor density allows us to capture these responses throughout the building and create a reasonable estimate for a client's transmission range.

Figure 5.9 shows the impact of overprotective APs on 802.11g clients in the network for the duration of the trace. It shows (1) the total number of overprotective APs that use protection mode unnecessarily, (2) the total number of active 802.11g clients associated with these APs, and (3) the total number of active 802.11g clients in the network. During busy periods of many active clients, the number of overprotective APs decreases as more 802.11b clients become active. Similarly, the number of 802.11g clients increases and, during these busy periods, 25–50% of them are associated with overprotective APs.

A more practical protection policy would provide two benefits to clients in the network. First, the 802.11g clients associated with overprotective APs could potentially improve their throughput substantially. With large frames transmitted at 54 Mbps without the need for CTS-to-self, these clients could potentially improve their throughput by

Figure 5.9: Overprotective APs and active 802.11g clients during the busy period of the trace.

a factor of two.² Of course, this result is an upper bound: not every 802.11g client would be able to transmit at full rate, and multiple clients would still contend for the channel. However, we have found that the network is rarely at maximum utilization, even during the busiest periods. As a result, 802.11g clients should be able to benefit, especially when performing bulk transfers and the wireless network is the bottleneck hop in their path.

Second, reducing the use of CTS-to-self reduces the possibility of exposed terminals in the network, which could improve the performance of the network. Like ARP and other low-rate short frames, CTS frames have relatively high penetration and can reserve the channel across a larger space than necessary when transmitting data frames at high rates.

²CTS: 248 μ s (our APs send CTS at 2 Mbps with the long preamble), SIFS: 16 μ s, MSS TCP at 54 Mbps: 248 μ s, SIFS: 16 μ s, ACK: 28 μ s, backoff (with g): 16/2*20, backoff (with b/g): 32/2*20. The potential performance improvement is $(248+16+248+16+28+32/2*20)/(248+16+28+16/2*20) = 1.98$.

5.6 TCP performance analyzer

The transport analyzer models TCP performance and behavior for interactive and bulk transfer flows. First we reconstruct per-flow TCP state to infer detail TCP events based on the analysis of [JID⁺07, RKS06], designed for wired passive monitors. The analyzer observes the TCP data sequence and ACK sequence to infer the causes of out-of-order events like fast or regular retransmission, spurious/unneeded retransmission, and packet reordering. In addition, since we have detailed information on the wireless transmissions (frame exchanges), we can further resolve some ambiguities. For example, a retransmission could be caused by the AP failing to deliver the TCP-DATA packet to the client, or the client failing to deliver the TCP-ACK packet to the AP. But if we have seen the client respond with an L2-ACK to the TCP-DATA packet to the AP, we can confirm that the client has received the packet at the TCP layer. Together with L2 information, the TCP analyzer can detect TCP-DATA and TCP-ACK losses at both sides.

Once we have reconstructed the flow characteristics, we can begin to diagnose any problems associated with the TCP flow. We first classify the flow as interactive or bulk transfer by checking the number of bytes sent by the client and server, and the number of full-sized TCP packets; bulk flows send substantially more bytes often with all but the last packet being full-sized. We discuss the bulk flow analysis after the next section.

5.6.1 Interactive TCP flow analysis

For interactive flows, we calculate the response time for a TCP data sequence segment to be acknowledged. Note that this time may not be the RTT during loss recovery as it may take the sender several round trips to finally get the ACK for the data; we mark a particular data segment as having a slow response time if it exceeds a certain threshold (*e.g.*, 200 ms for interactive connections [res]) and inspect the delay and loss characteristics during that period. If most losses happen at the Internet side, the ana-

Figure 5.10: Response time and TCP behavior of a user who reported substantial delay with an interactive SSH session.

lyzer diagnoses the connection as having Internet side problems and terminates, because it does not have further information to proceed.

If the client has wireless losses, the TCP analyzer checks for broadband interference from microwaves or co-channel interference from other clients in the nearby area from the broadband interference and media access modules. It also checks if the failed frame exchange is caused by loss of L2-DATA or L2-ACK transmissions. For instance, we have observed clients who suffer heavy AP to client L2-ACK losses. Although these losses would not result in a loss to higher layers since the DATA has been delivered, the client would retry excessively because it cannot receive the L2-ACKs from its AP. The excessive retries would backlog the client's TCP-ACKs, eventually causing the server to timeout and retransmit (spuriously).

If the client does not have wireless losses but instead suffers from high link delay, the TCP analyzer returns the major cause of the link delay. The cause could be contention from the other clients, contention from other connections from the same client itself, or contention due to excessive backoff and retries [CAV⁺07b].

As an example of the operation of the TCP analyzer, Figure 5.10 shows a time series graph of the response time and TCP behavior of a user reporting a slow SSH connection during a lab seminar. The graph shows the user suffers from long response times (200 ms – 2.5 seconds) even though the connection is to a server in the same department.

The TCP analyzer reports high losses from the user's client device to the AP (shown as gray crosses), meaning the client's TCP-ACK or TCP-DATA packets do not get through. It finds that most server retransmissions (shown as black diamonds) are not fast retransmissions, but caused by TCP timeout. Therefore the retransmissions usually occur after 200 ms (WindowsXP default [RKS06]). Further, often the TCP-ACK of the retransmission is also lost, causing the SSH server to exponentially backoff the RTO timer.

The analyzer does not find microwave or hidden terminal events, and continues to look for other causes. It analyzes the rate used by the wireless transmissions, and finds that the client has 10-times higher losses for 802.11g transmissions than 802.11b transmissions. The TCP analyzer concludes the final diagnosis as high client losses caused by a poor rate adaption algorithm.

Since the user typically does not indicate the specific connections that are slow, the TCP analyzer performs diagnosis on all connections from the user. Based on the diagnosis from each connection, the analyzer returns the major cause across all diagnosis reports. We have noticed that the diagnoses across active simultaneous connections from the same user are very consistent (except server-side Internet losses). More surprisingly, the analyzer reports bursts of server or client-side Internet losses across different clients. While Internet losses should be more dependent on the Internet paths and the end server, these results may indicate that our wireless gateway is dropping packets. We are still investigating this issue with our campus network operations staff.

5.6.2 Bulk TCP flow analysis

Next we describe how we can use the media access model as a basis for diagnosing problems with TCP throughput for wireless users, and show that there can be many causes that can limit TCP throughput. Given a TCP flow using wireless, we first identify whether throughput is the critical bottleneck of a TCP flow. We then examine the flow and determine whether throughput performance appears to be limited by wireless network conditions. If so, we then use the media access model to determine critical

path delays for packets in the flow, evaluate how those delays interact with TCP, and assign a root cause for why the TCP flow was limited when using the network.

The first step is to determine whether a TCP flow contained data transfer periods whose throughput could be limited by wireless conditions. Since a given TCP flow may have idle periods (e.g., think times during persistent HTTP connections), we identify periods of time during a TCP flow when it is performing a bulk data transfer. We call such a period a TCP transaction. A TCP transaction period starts when we observe new, unacknowledged TCP data and ends when all outstanding data packets generate an acknowledgment. Most of the packets in this period must also be MSS-sized except for the last data packet, reflecting a period when a bulk of data is being sent. We then calculate the amount of data transferred during the flow to identify flows of sufficient size that they could potentially take full advantage of the wireless channel; we currently use a threshold of 150 KB.

We then take these flows and determine whether throughput performance appears to be limited by wireless network conditions, and, if so, why. In our approach, we assume that there is a single root cause and that factors are largely independent (e.g., wireless loss is independent of Internet loss). We then analyze the flow through a series of filters. First, if the flow is achieving near optimal throughput for the rate used, we label it ideal and perform no further analysis. If the flow announces a zero receiver window, we label it as receiver window limited.

We then determine if the Internet part of the connection was the bottleneck. To do this, we estimate what the TCP throughput for the flow *might have been* if using the wireless network under ideal conditions (no wireless loss, no contention, no wireless RTT, etc.). We use Padhye's TCP throughput estimation analysis [PFTK00] to perform this estimation, calculating idealized throughput just using the measured Internet RTT, measured Internet loss rate, and an estimated RTO. If the estimated Internet throughput is close to measured, we label the flow as Internet limited.

We then examine wireless losses and add wireless loss rate into the throughput estimation; if throughput drops substantially, we label the flow as wireless loss limited.

Figure 5.11: Root wireless causes that limit TCP flow throughput.

At this point, remaining flows are usually victims of high wireless RTTs. Using the media access model, if the AP-to-client delay is larger than client-to-AP, we label the flow as being either limited by queuing delays (d_q) (background traffic, frames to self, or frames to other clients), power-save delays (d_{ps}), or DCF transmission (d_{dcf}). If the primary delay is DCF, and the transmission time only takes less than half of the DCF delay, we label it as contention limited. Otherwise, we check whether the client potentially could have used a higher rate; if so, we label it as rate limited. If the client is using an 802.11g rate and the AP is in protection mode, we estimate the potential benefits of removing the CTS-to-self overhead [CBB⁺06]; if this benefit is higher compared with using protection mode at the highest 802.11g rate, we label the flow as protection-mode limited.

What are the sources of wireless behavior that impact TCP throughput in a

production network? We apply the above analysis to all measured TCP flows on the building wireless network for 24 hours on a typical weekday. Figure 5.11 shows the breakdown of root causes across flows identified as bulk data transfer flows.

The graph shows four interesting results. First, flows can be limited by a wide range of different causes; for a particular user experiencing poor TCP throughput, we must model and check all such causes to diagnose their particular problem. Second, over 25% of the flows are limited by the faulty 802.11g link-level retry policy used by the APs in the building. At 802.11g rates, the APs only perform one link-level retry before giving up when the AP is in protection mode; not surprisingly, this policy limits TCP performance when using those rates. Third, over 30% of the flows are limited by the use of 802.11g protection mode. Fourth, nearly 30% of the flows turn out to be limited by the receiver window size — indicating that although wireless conditions may initially be suspect, throughput can be limited simply by the TCP stack configuration. Any diagnosis system must suspect causes outside of wireless as well.

5.7 Shaman diagnostic tools

This section describes the two Shaman diagnostic tools, one designed for users to invoke on demand about wireless problems that they are experiencing and a second designed to alert network administrators about pervasive network problems.

5.7.1 User diagnostic tool

The goal of the Shaman user diagnostic tool is to answer queries on demand from users about performance problems that they are experiencing with the enterprise wireless network. The user interface to the tool is a simple Web form with fields for identifying a user's network device, an approximate time that the problem occurred, a contact email address, and any notes about the problem for archiving. Figure 5.12 shows a screened of a form filled in to report the DHCP behavior.

When a user submits the form, the tool executes on a backend server with

The screenshot shows a web browser window titled "Wireless Problem Report - Mozilla Firefox". The address bar shows "http://anonymized". The form contains the following elements:

- I've had wireless problems within:** Radio buttons for "last minute" (selected), "last ten minutes", and "other (specify in the problem description)".
- I am browsing this page using:** Radio buttons for "same computer" (selected) and "another (* specify MAC address in problem description)".
- Problem Description:** A text area containing the text "Laptop won't get on the interwebs."
- Location/Room:** A text input field containing "lobby".
- Name:** A text input field containing "Disgruntled User".
- Email:** A text input field containing "disgruntled@yahoo.com".
- SUBMIT:** A button at the bottom of the form.

At the bottom of the browser window, there is a "Find:" search bar and buttons for "Find Next", "Find Previous", "Highlight all", and a checkbox for "Match case".

Figure 5.12: Shaman problem report form.

the fields from the form as input, invokes the various network analyzers on the trace covering the time of the reported problem, evaluates the output of the analyzers, and makes a decision about the primary cause of the problem. The tool then reports this decision back to the user, accompanied with additional results for context if the user is interested in more details. Figure 5.13 shows a screen shot of results reported from the tool for the user experiencing the DHCP problem described in [CAV⁺07b]. The tool correctly identifies DHCP as the main problem.

The decision algorithm of the user diagnostic tool works as follows. First it examines the wireless loss rates returned by the various analyzers, including both mobility operations (e.g., DHCP or association) and during network communication (e.g., media access and TCP). If the loss rates are high, it uses the broadband analyzer to determine whether a microwave is active during that period. If so, it reports the microwave as the dominant cause.

Otherwise it compares the results of the various analyzers in more detail to determine a dominant cause. As a common basis for comparison, it uses a time duration metric for each analyzer that reports a problem. We chose time as the common metric because it intuitively matches how users tolerate and react to network problems;

Figure 5.13: Shaman basic diagnosis.

when a problem persists long enough to be affect the user network experience, users are sufficiently motivated to invoke a service to diagnose the problem.

For the mobility protocols, this duration is the critical time spent in the various protocols (association, DHCP, ARP). For TCP, this duration is the total time during which TCP was under performing (low throughput or high delay). The tool assigns the dominant cause for the poor user experience to the analyzer with the longest time metric. The tool then reports this time, which analyzer reported the cause, and any detailed explanation returned by the analyzer (e.g., excessive AP disassociation and re-association, timeouts while requesting DHCP leases, contention constraining TCP throughput, losses or delays causing TCP timeouts, etc.).

5.7.2 Network alert tool

The goal of the Shaman network alert tool is to pro-actively report serious pervasive problems to network administrators. Serious pervasive problems are those

that simultaneously affect multiple clients at one or more APs and require the intervention of a network administrator to correct. Typically these problems are due to failures of critical network components (DHCP or DNS servers, routers, wireless management gateways, etc.), or persistent performance issues (e.g., poor coverage within a building [BCP⁺06]).

The design of the network alert tool includes analyzers that mirror the network analyzers described in previous sections. These alert analyzers operate continuously by invoking the network analyzers to detect pervasive problems. The tool counts the fraction of the client population affected by a specific type of event (e.g., clients performing DHCP requests who timeout). If this count is significant (we currently use a threshold of 75%), the tool triggers an alert by sending email to a wireless administration mailing list detailing the event, results from the analyzers, and the clients affected.

Motivated by experience, we have currently implemented one alert module for DHCP. The majority of our pervasive problems in which multiple users have simultaneously submitted wireless reports that all have the same underlying cause have been linked to DHCP — the DHCP server itself has failed, the router between the wireless distribution network and the DHCP server was overwhelmed by a denial-of-service attack, etc. Specifically, our DHCP analyzer informs the alert tool of clients that remain in the **Requesting** state for more than two seconds. As an example, our tool produced an alert for a particular day around noon. The alert lasted for twelve minutes, during which 6–10 clients were trying to obtain new DHCP leases. 83% percent of the requesting clients were unsuccessful. The clients were associated with different APs, indicating a network-wide DHCP problem. Further investigation revealed that the DHCP server was unreachable from parts of the wireless network.

5.8 Acknowledgement

Chapter 5, in part, is a reprint of the material as it appears in the SIGCOMM Conference, 2006, Cheng, Yu-Chung; Bellardo, John; Benko, Peter; Snoeren, Alex C.;

Voelker, Geoffrey M.; Savage, Stefan. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part, is a reprint of the material as it appears in the SIGCOMM Conference, 2007, Cheng, Yu-Chung; Afanasyev, Mikhail; Verkaik, Patrick; Benko, Peter; Chiang, Jennifer; Snoeren, Alex C.; Voelker, Geoffrey M.; Savage, Stefan. The dissertation author was the primary investigator and author of this paper.

Chapter 6

Conclusion

Modern enterprise networks are of sufficient complexity that even *simple* faults can be difficult to diagnose — let alone transient outages or service degradations. Nowhere is this problem more apparent than in the 802.11-based wireless access networks now ubiquitous in the enterprise. We believe that such diagnosis must be automated, and that networks must eventually address transient failure without human involvement.

In this dissertation, we have described Shaman, a system for performing comprehensive and automatic diagnosis on wireless problems down to fine-grained low-level root causes. Shaman combines wireless monitoring infrastructure, trace synchronization, a collection of network analyzers, and two diagnostic tools into a single diagnostic system for enterprise wireless networks.

To demonstrate our approach, we have deployed a large-scale instance of Shaman using 193 radios in our department building and used a 24-hour trace captured by our monitoring infrastructure to demonstrate complex interactions such as co-channel interference that would otherwise be difficult to analyze. We have described the algorithms used to synchronize traces, unify common frames, and reconstruct the link- and transport-layer conversations embedded in those frames. We also present algorithms that accurately model the impact of protocol behavior from the physical layer to the transport layer. While some sources of delay can be directly measured, many of the delay

components, such as AP queuing, back-offs, contention, etc., must be inferred. To infer these delays from measurements, we develop a detailed model of MAC protocol behavior, both as it is described in the 802.11 specification as well as how it is implemented in vendor hardware. We find that no one anomaly, failure or interaction is singularly responsible for these issues and that a holistic analysis may in fact be necessary to cover the range of problems experienced in real networks.

For more widespread deployment, we envision migrating the monitoring infrastructure used by Shaman into the access points themselves. Merging this functionality into the APs reduces the deployment cost and simplifies analysis, albeit reducing monitoring coverage. Exploring this evolution in wireless monitoring and automated diagnosis remains an exciting open problem. For those interested in using or contributing to our efforts, the Shaman hardware specification and software are available for download at <http://sysnet.ucsd.edu/wireless/>.

Bibliography

- [ABCQ04] Atul Adya, Paramvir Bahl, Ranveer Chandra, and Lili Qiu. Architecture and techniques for diagnosing faults in IEEE 802.11 infrastructure networks. In *Proceedings of Mobicom*, 2004.
- [aira] Air magnet. <http://www.airmagnet.com>.
- [airb] Airwave. <http://www.airwave.com>.
- [BC00] Paul Barford and Mark Crovella. Critical path analysis of TCP transactions. In *Proceedings of SIGCOMM*, Stockholm, Sweden, 2000.
- [BC03] Magdalena Balazinska and Paul Castro. Characterizing mobility and network usage in a corporate wireless local-area network. In *Proceedings of USENIX MobiSys*, 2003.
- [BCP⁺06] Paramvir Bahl, Ranveer Chandra, Jitendra Padhye, Lenin Ravindranath, Manpreet Singh, Alec Wolman, and Brian Zill. Dair: A framework for managing enterprise wireless networks using desktop infrastructure. In *Proceedings of MobiSys*, 2006.
- [Bel06] John Bellardo. *Secure and Efficient Wireless Networks*. PhD dissertation, University of California, 2006.
- [BVBR02] Anand Balachandran, Geoffrey M. Voelker, Paramvir Bahl, and P. Venkat Rangan. Characterizing user behavior and network performance in a public wireless LAN. In *Proceedings of SIGMETRICS*, 2002.
- [Cam05] Campus Computing. 2005 National Survey of Information Technology in U.S. Higher Education, 2005.
- [CAV⁺07a] Yu-Chung Cheng, Mikhail Afanasyev, Patrick Verkaik, Peter Benko, Jennifer Chiang, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Automating cross-layer diagnosis of enterprise wireless networks. In *Proceedings of the ACM SIGCOMM Conference*, Kyoto, Japan, August 2007.
- [CAV⁺07b] Yu-Chung Cheng, Mikhail Afanasyev, Patrick Verkaik, Jennifer Chiang, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. The automatic 802.11 wireless diagnosis system. In *Submitted for publication*, 2007.

- [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 *Proceedings of the ACM SIGCOMM Conference*, Pisa, Italy, September 2006.
- [CBV05] P. Chatzimisios, A. C. Boucouvalas, and V. Vitsas. Performance analysis of the IEEE 802.11 MAC protocol for wireless LANs. *Wiley International Journal of Communication Systems*, 18(6):545–569, June 2005.
- [Chi06] Jennifer Chiang. Analysis of broadband interference in enterprise 802.11 wireless networks. Masters dissertation, University of California, 2006.
- [Com99] IEEE Computer Society LAN MAN Standards Committee. IEEE standard 802.11, wireless LAN media access control (mac) and physical layer (phy) specifications, 1999.
- [CPWZ07] Ranveer Chandra, Jitendra Padhye, Alec Wolman, and Brian Zill. Where are they and what are they doing: On the importance of locating clients for managing enterprise WLANs. In *Proceedings of NSDI*, 2007.
- [CPZ06] Ranveer Chandra, Venkata Padmanabhan, and Ming Zhang. Wifiprofiler: Cooperative diagnosis in wireless lans. In *Proceedings of MobiSys*, 2006.
- [Dal05] Ellen Daley. Enterprise LAN Grows Up, 2005. <http://www2.cio.com/analyst/report3401.html>.
- [Del06] Dell’Oro Group. Wireless LAN Five Year Forecast Report, 2006.
- [DR92] Dan Duchamp and Neil F. Reynolds. Measured performance of a wireless LAN. In *Proceedings of IEEE LCN Conference*, 1992.
- [EGE02] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *Proceedings of OSDI*, 2002.
- [ES96] David Eckardt and Peter Steenkiste. Measurement and analysis of the error characteristics of an in-building wireless network. In *Proceedings of SIGCOMM*, 1996.
- [Gar05] Gartner. Market Share: Wireless LAN Equipment Worldwide, 2005.
- [GBG06] K. N. Gopinath, Pravin Bhagwat, and K. Gopinath. An empirical analysis of heterogeneity in IEEE 802.11 MAC protocol implementations and its implications. In *Proceedings of WiNTECH 2006*, 2006.
- [HCP05] Félix Hernández-Campos and Maria Papadopouli. A comparative measurement study of the workload of wireless access points in campus networks. In *Proceedings of PIMRC*, 2005.

- [HKA04] Tristan Henderson, David Kotz, and Ilya Abyzov. The changing usage of a mature campus-wide wireless network. In *Proceedings of Mobicom*, 2004.
- [HRABR04] Camden C. Ho, Krishna N. Ramachandran, Kevin C. Almeroth, and Elizabeth M. Belding-Royer. A scalable framework for wireless network monitoring. In *Proceedings of the Workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, 2004.
- [JID⁺07] S Jaiswal, G Iannaccone, C Diot, J Kurose, and D Towsley. Measurement and classification of out-of-sequence packets in a tier-1 IP backbone. *IEEE/ACM Transactions on Networking*, 15(1), 2007.
- [JPS04] Jangeun Jun, Pushkin Peddabachagari, and Mihail Sichitiu. Theoretical maximum throughput of IEEE 802.11 and its applications. In *Proceedings of the 2nd IEEE International Symposium on Network Computing and Applications*, 2004.
- [JRABR05] Amit P. Jardosh, Krishna N. Ramachandran, Kevin C. Almeroth, and Elizabeth M. Belding-Royer. Understanding congestion in IEEE 802.11b wireless networks. In *Proceedings of IMC*, 2005.
- [KE02] David Kotz and Kobby Essien. Analysis of a campus-wide wireless network. In *Proceedings of Mobicom*, 2002.
- [KEES03] R. Karp, J. Elson, D. Estrin, and S. Shenker. Optimal and global time synchronization in sensor networks. Technical Report CENS-TR0012, CENS, UCLA, 2003.
- [kis] Kismet. <http://www.kismetwireless.net>.
- [KR94] H Yang KK Ramakrishnan. The ethernet capture effect: Analysis and solution. In *Proceedings of 19th Conference of Local Computer Networks*, 1994.
- [KSM05] Byung-Jae Kwak, Nah-Oak Song, and Leonard E. Miller. Performance analysis of exponential backoff. *IEEE/ACM Transactions on Networking*, 13(2), April 2005.
- [KVSA04] A. Kochut, A. Vasan, A. U. Shankar, and A. Agrawala. Sniffing out the correct physical layer capture model in 802.11b. In *Proceedings of ICNP*, 2004.
- [mad03] Madwifi driver for linux, 2003. <http://sourceforge.net/projects/madwifi/>.
- [MRWZ06] Ratul Mahajan, Maya Rodrig, David Wetherall, and John Zahorjan. Analyzing the MAC-level behavior of wireless networks in the wild. In *Proceedings of SIGCOMM*, 2006.

- [MV05] Marvin McNett and Geoffrey M. Voelker. Access and mobility of wireless pda users. *Mobile Computing and Communications Review (MC2R)*, 9(2), April 2005.
- [NKNS96] Giao T. Nguyen, Randy H. Katz, Brian Noble, and Mahadev Satyanarayanan. A trace-based approach for modeling wireless channel behavior. In *Winter Simulation Conference*, 1996.
- [PFTK00] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP reno performance: A simple model and its empirical validation. *IEEE/ACM Transactions on Networking*, 2000.
- [QBRZ06] Lili Qiu, Paramvir Bahl, Ananth Rao, and Lidong Zhou. Troubleshooting wireless mesh networks. *ACM SIGCOMM Computer Communication Review*, October 2006.
- [RBRA04] Krishna N. Ramachandran, Elizabeth M. Belding-Royer, and Kevin C. Almeroth. Damon: A distributed architecture for monitoring multi-hop mobile networks. In *Proceedings of SECON*, 2004.
- [res] Rspnse time overview. <http://www.useit.com/papers/responsetime.html>.
- [RHA04] Maxim Raya, Jean-Pierre Hubaux, and Imad Aad. Domino: a system to detect greedy behavior in IEEE 802.11 hotspots. In *Proceedings of MobiCom*, 2004.
- [RKS06] Sushant Rewaskar, Jasleen Kaur, and F. Donelson Smith. A passive state-machine approach for accurate analysis of TCP out-of-sequence segments. *ACM SIGCOMM Computer Communication Review*, 36(3), 2006.
- [RRM⁺05] Maya Rodrig, Charles Reis, Ratul Mahajan, David Wetherall, and John Zahorjan. Measurement-based characterization of 802.11 in a hotspot setting. In *Proceedings of ACM E-WIND*, 2005.
- [SB04] David Schwab and Rick Bunt. Characterising the use of a campus wireless network. In *Proceedings of Infocom*, 2004.
- [SDG⁺06] Anmol Sheth, Christian Doerr, Dirk Grunwald, Richard Han, and Douglas Sicker. Mojo: A distributed physical layer anomaly detection system for 802.11 WLANs. In *Proceedings of MobiSys*, 2006.
- [TB00] Diane Tang and Mary Baker. Analysis of a local-area wireless network. In *Proceedings of Mobicom*, 2000.
- [WKHW02] Andreas Willig, Martin Kubisch, Christian Hoene, and Adam Wolisz. Measurements of a wireless link in an industrial environment using an IEEE 802.11-compliant physical layer. *IEEE Transactions on Industrial Electronics*, 43(6), 2002.

- [YYA04] J Yeo, M Youssef, and A Agrawala. A framework for wireless lan monitoring and its applications. In *Proceedings of ACM WiSe*, 2004.
- [YYHA05] Jihwang Yeo, Moustafa Youssef, Tristan Henderson, and Ashok Agrawala. An accurate technique for measuring the wireless side of wireless networks. In *Proceedings of WiTMeMo*, 2005.