

Automating Cross-Layer Diagnosis of Enterprise Wireless Networks

Yu-Chung Cheng, Mikhail Afanasyev, Patrick Verkaik, Péter Benkő[†], Jennifer Chiang, Alex C. Snoeren, Stefan Savage, and Geoffrey M. Voelker

Department of Computer Science and Engineering
University of California, San Diego

[†]Traffic Analysis and Network Performance Laboratory (TrafficLab)
Ericsson Research, Budapest, Hungary

ABSTRACT

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. 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 paper, we present a set of modeling techniques for automatically characterizing the source of such problems. In particular, we focus on data transfer delays unique to 802.11 networks — media access dynamics and mobility management latency. Through a combination of measurement, inference and modeling we reconstruct sources of delay — from the physical layer to the transport layer — as well as the interactions among them. We demonstrate our approach using comprehensive traces of wireless activity in the UCSD Computer Science building.

Categories and Subject Descriptors

C.4 [Performance of Systems]: Modeling techniques

General Terms

Modeling, Measurement, Performance

Keywords

Wireless networks, 802.11, modeling, measurement

1. INTRODUCTION

“Is your wireless working?” The familiarity of this refrain underscores both our increasing dependence on ubiquitous Internet connectivity and the practical challenges in delivering on this promise. The combination of unlicensed spectrum and cheap 802.11 silicon have driven a massive deployment of wireless access capability — which started in the home and was soon followed by the workplace. Today over two-thirds of U.S. corporations provide WiFi-based untethered Internet connectivity [8]. However, 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 a radio-frequency (RF) site survey and, ideally, managed to minimize contention, maximize throughput, and 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 variability, dynamic addressing and authorization, VLAN setup, as well as the myriad complexities of the wired network itself.

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 ten 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 introduce further intricacies that are unique and qualitatively harder to diagnose.

Among these issues, 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, VLANs, ARP, DHCP and often proprietary mobility management and authentication systems. Unsurprisingly, the resulting Rube-Goldberg contraption has its own unique failure modes.

In practice, few network administrators have both the detailed visibility into network behavior and the breadth of knowledge needed to diagnose such problems. When they do, the process is highly labor intensive and rarely cost effective except for the most severe and persistent problems. Even then, the range of interactions and lack of visibility into their causes may stymie manual diagnosis. In one recent episode at UCSD, 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.¹

¹We believe we have diagnosed their problem — a subtle bug in

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to publish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCOMM’07, August 27–31, 2007, Kyoto, Japan.

Copyright 2007 ACM 978-1-59593-713-1/07/0008 ...\$5.00.

Our experience suggests that such diagnoses must be fully automated to be effective. As a first step in this direction, we have developed models of wireless delays from the physical layer to the transport layer. In particular, we demonstrate techniques to infer the causes and effects of both link-layer delays and mobility management delays. To demonstrate their effectiveness, we use our models to investigate the causes of transient performance problems from traces of wireless traffic in the four-story UCSD Computer Science building. We find that no one anomaly, failure or interaction is singularly responsible for these issues in our environment — suggesting that our holistic analysis approach may be necessary to cover the range of problems experienced in real networks.

The remainder of this paper is structured as follows. We first review the literature we build upon and related efforts in Section 2. In Section 3 we summarize the monitoring system we use to collect trace data for use with our models. We then outline the many potential sources of service disruption — the gauntlet faced by each 802.11 packet — in Section 4. Sections 5 and 6 describe our techniques for modeling media access and mobility management behavior, respectively, including an analysis of the problems identified at our location, followed by our conclusions.

2. RELATED WORK

Ever since wireless local-area networks such as 802.11 have been deployed, researchers have sought to understand how these systems behave and perform based on empirical observations. The monitoring systems used to make these observations have increasingly expanded both in complexity and scope over time. Early systems used existing infrastructure, such as the wired distribution network and the APs, to record wireless traffic and network characteristics [2, 16]. Later systems deployed small numbers of dedicated monitoring nodes, sometimes concentrated near the APs, other times distributed throughout the network, thereby pushing the frontier of observation into the link-layer domain [12, 18, 23]. Recent efforts have substantially scaled monitoring platforms to observe large, densely deployed networks in their entirety [1, 7], providing the ability to observe every link-layer network transmission across location, frequency, and time [7].

These monitoring systems have been used to directly measure a number of interesting aspects of 802.11 behavior and performance, ranging from application workloads and user mobility at the high level to wireless loss, rate adaptation, and handoff delay at the link layer [10, 13, 19] and even physical layer anomalies [22].

Other techniques infer more detailed network events and characteristics, such as link-layer loss and the transmitters of packets lacking MAC addresses [4, 7, 18], co-channel interference and overprotective APs [7], misbehaving and heterogeneous devices [4, 7, 9], root causes of physical-layer anomalies [22], and regions of poor coverage [4]. We greatly expand upon these detailed efforts and present techniques to infer critical path delays [3] 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 (Section 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 [6], the maximum throughput

the AP vendor’s implementation — but it is easy to understand in retrospect why its discovery was challenging to find through trial and error.

Start	1/11/07 @ 00:00
Duration	24 hours
Radio Monitors	192
Infrastructure APs	40
Wireless clients	188
Raw trace size	96 GB
Unique frames captured	210 M

Table 1: Summary of trace characteristics.

of a flow in an 802.11 network [14], and the saturation throughput and expected access delay of contending nodes [17]. 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 interference, 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.

Three recent systems are closely related to the goals of this paper. DAIR uses wireless USB dongles attached to desktop machines in an enterprise wireless network to measure wireless events throughout the network [1]. DAIR applications install filters at the wireless monitors to trace information of interest and store it in a central database; applications (inference engines) then use this data to perform analyses. Very recent work on DAIR develops management applications that take advantage of client location, such as identifying regions in the network experiencing consistently poor coverage [4]. Our goals are similar in that we develop analyses to aid network management, but we base analysis on a global understanding of network behavior across all protocol layers.

WiFiProfiler [5] also helps users troubleshoot wireless connectivity problems. WiFiProfiler relies upon peer diagnosis among clients without the involvement of system administrators, while our paper uses 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. 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 [22]. While we are interested in physical-layer issues, we identify them as just one cause among many across the interacting protocol layers.

3. TRACE COLLECTION

The modeling approach we describe in this paper operates on detailed traces of wireless activity. We use the Jigsaw system described in [7] to collect the traces we use in this paper, and we briefly sketch the system in this section. Jigsaw is a distributed wireless monitoring platform that we have deployed in our department building to monitor the production 802.11 network. The production 802.11 network consists of 40 Avaya AP-8 802.11 b/g access points covering four floors and the basement. The APs are identically configured (except for their channel assignment) and support both 802.11b and 802.11g without encryption.

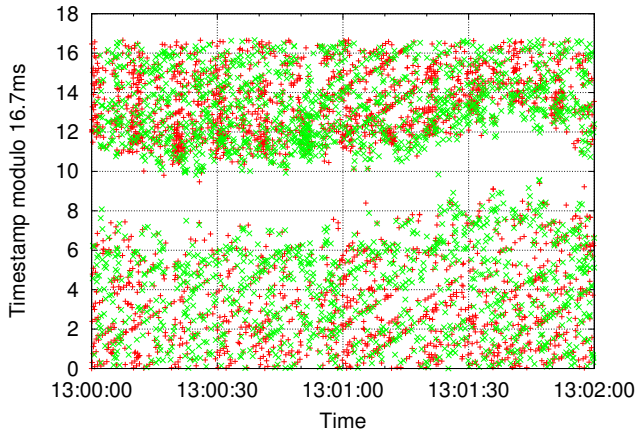


Figure 1: Physical error frame pattern during microwave oven use. The y-axis depicts (time % 16) ms, showing the offset of physical error packets within the 16 ms microwave period. The x-axis shows two minutes of microwave use.

The hardware monitors consist of 192 radios interspersed between the infrastructure APs. The radios passively monitor the wireless network and report all wireless events across location, channel, and time via a private wired network to a back-end storage server. Jigsaw merges and time synchronizes these separate radio traces into a single, global unified trace. Moreover, Jigsaw performs this operation in real time; a single 2.2GHz AMD Opteron server can synchronize one minute of raw trace data in under 15 seconds.

We currently configure Jigsaw to capture the first 120 bytes of each wireless frame. As a result, the aggregate monitor traffic from all radios ranges from 2–10Mbps and is roughly five times the amount of production wireless traffic. On a typical weekday, the raw traces total 80 GB and the merged, synchronized trace is roughly 10 GB in size. Our analyses are conducted on the synchronized trace; the raw traces are used for debugging purposes only. The particular trace used in this paper was collected on Thursday, January 11, 2007. Table 1 shows various high-level characteristics of the users and traffic contained in this trace.

4. 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.

4.1 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 1 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. 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* [15], which means a radio often decodes the frame with higher signal strength when two packets collide at a receiver.

4.2 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 μ s, although 802.11g permits the use of a “short” 9- μ s slot time under certain circumstances). After the backoff it must sample the channel for the duration of a “DIFS” interval (50 μ 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 man-

agement. 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 our previous 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. [7]

4.3 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.

4.4 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 this paper to systematize the analysis of these issues to better understand the source of such transient problems.

5. MEDIA ACCESS MODEL

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 2 summarizes the definitions of the timestamps and delays in our model, and Figure 2 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 trans-

mission behavior necessary to perform the inferences, represent a key contribution of this paper.

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.

5.1 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 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 flushes the appropriate packets from the power-save queue into the transmit queue when 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

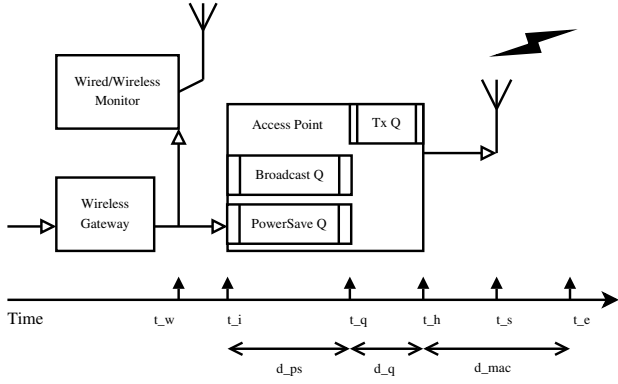


Figure 2: Representation of wired distribution network, queuing behavior in the AP, and frame transmission. The arrows indicate where in the network we measure and infer timestamps as frames traverse the network, and the corresponding delays we calculate.

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

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

Table 2: Summary of timestamps and delays determined by the media access model.

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.

5.2 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}).

5.3 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 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.

```

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 3: Simplified 802.11 DCF operation for unicast.

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 4 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 3. This code segment is a simplified version of the unicast DCF operation in the 802.11 standard [11]. 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 4 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 ($X = 50 + 15 \cdot 20$) 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 $547 + 15 \cdot 20 = 847\mu s$. 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 3.

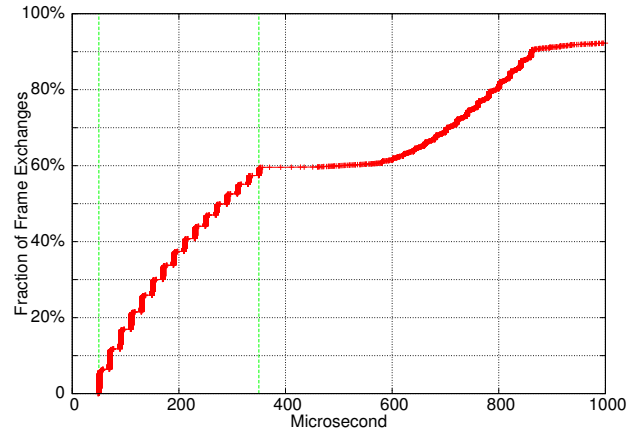


Figure 4: Access delay (d_{t0}) distribution of one hour of head-of-line blocked frame exchanges from an Avaya AP-8 AP to a client doing a bulk TCP download.

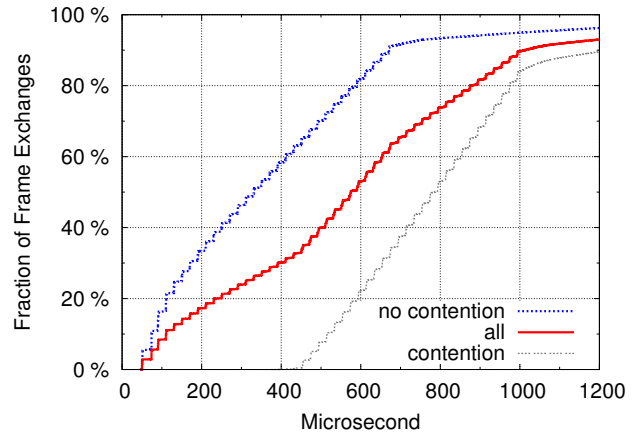


Figure 5: Access delay (d_{t0}) distribution of head-of-line blocked frame exchanges from a Cisco Aironet 350 AP to a client doing a bulk scp download.

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 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.

At first glance the distribution looks dramatically different from the Avaya’s distribution in Figure 4. 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

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. The line forms a set of stairs starting from around $400 \mu\text{s}$. 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 4.

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

In summary, even though we must infer the timing of some of the events that 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 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 6 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 TCP throughput

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 the TCP flow is attempting to transmit and maximum speed. We then examine the flow to determine whether throughput performance appears to be limited by wireless network conditions. If so, we 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 throughput was limited when using the wireless network. Our goal is to show the systems administrator the distribution of potential performance problems so they can focus on improving the major bottlenecks.

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 have been acknowledged. 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 calculate the amount of data transferred during the flow to identify transactions 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 transactions 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 transaction through a series of filters. First, if the transaction is achieving near optimal throughput for the 802.11 rate used, we label it ideal and perform no further analysis. Additionally, if the client ever announces a zero-sized receiver window during the bulk transfer, we label it as receiver window limited.

Next, we model the TCP throughput by extracting the network and host conditions. We use TCP throughput estimation analysis [20] to perform this estimation, calculating idealized throughput from the measured path RTT, measured path loss rate, and an estimated RTO. We fine-tune the throughput model by identifying the client OS from their DHCP messages and applying OS dependent TCP parameters [21]. To ensure the throughput model works for a particular transaction, we compare the modeled throughput with the actual throughput measured. We proceed only if the modeled throughput is within 10% of the actual throughput. To determine if the wired portion of the connection is the bottleneck, we estimate what the TCP throughput for the transaction *might have been* if the client was directly connected to the Ethernet by removing the wireless losses and wireless RTT. If the estimated throughput improves by more than 20% compared to the measured throughput, we label the transaction as Internet limited.

If the Internet is not the limit, we examine if wireless loss exported to the TCP layer is the root cause by adding wireless loss

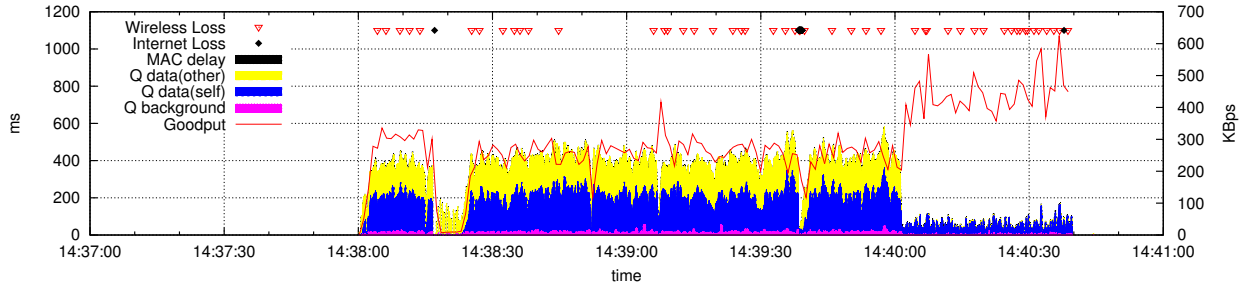


Figure 6: Critical path delays, goodput, and losses over time for frame exchanges from an AP to a client doing a bulk TCP download.

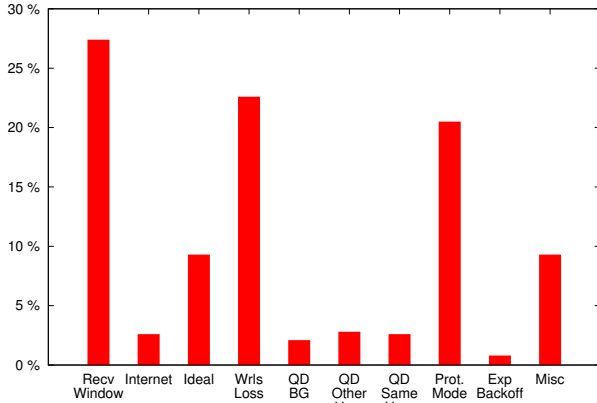


Figure 7: Root causes that limit TCP flow throughput.

rate into the throughput estimation; if throughput drops over 20%, we label the flow as wireless-loss limited. At this point, the remaining flows are likely victims of the high delays in the wireless network. We label the transactions as being either limited by queuing delays (d_q , i.e., background traffic, frames to self, or frames to other user), power-save (d_{ps}), or MAC delay (d_{mac}) accordingly.

We further investigate transactions limited by high MAC delays. If over half of the delay is contributed by exponential backoffs, we label the transaction as contention/backoff limited. Note while high wireless losses normally cause many exponential backoffs due to local retransmissions, we have already filtered out these cases. Typically transactions that are limited by backoffs have many local 802.11 retransmissions but do not suffer from TCP layer losses. For example, some vendors retry up to 15 times. The frame exchange might eventually go through but the exponential backoffs could take over 220 ms.

Finally, if an 802.11g client is connected to an over-protected AP using 802.11g protection mode [7], we label the transaction as over-protection limited. Over-protection happens when an idle slow 802.11b client causes all fast 802.11g clients to take up to two times longer to transmit data than in normal 802.11g. If none of these cases apply, we label the transactions as limited by some unknown factor. Such transactions likely would not benefit significantly by addressing any of the potential issues we considered.

We apply the above analysis to the bulk flows in a trace of typical wireless activity in our network (Section 3). we identify 2,605 bulk transactions in 1,375 TCP flows from 864 users. 85% of the flows are HTTP or SCP downloads. We filtered out 28% of the transactions because the modeled throughput is not within 10% of the measured throughput; we find, for example, that the analytic model over-predicts throughput for low-bandwidth flows, and we are continuing to refine our approach for these cases.

Figure 7 shows the breakdown of root causes across the remaining, properly-modeled transactions. 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 20% of the transactions are limited by wireless loss. This is mainly caused by a faulty 802.11g link-level retry policy used by the APs in our 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 22% of the transactions are limited because our APs are too conservative in using 802.11g protection mode. Thus, both the retry and over-protection problems can be easily solved by simply revising the AP software. Indeed, the vendor has acknowledged the problem we reported and started to investigate the solutions. Finally, nearly 27% of the transactions 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 client’s TCP stack configuration. Any diagnosis must suspect causes outside of wireless as well.

6. MOBILITY

The second class of overhead in the 802.11 environment is the expense of the various types of mobility management, including scanning for access points, association, ARP, DHCP, authentication, etc. Mobility management overhead can cause delays, for example, while waiting for an IP address when joining a network for the first time, or because of interruptions of normal network activity due to scanning for alternate APs. In this section we describe how we model delays due to mobility management overhead, and we apply the model to traffic in our trace to illustrate how these delays can impact network use under typical wireless conditions.

6.1 Overhead analysis

We categorize mobility management packets into one of eight categories: scanning, PSM sleep, association (including authorization, association, reauthorization, and reassociation requests), DHCP, DNS, ARP, TCP, and “misc”. In our environment, “misc” includes WEP/WPA (while none of our APs support encryption, clients may occasionally send such packets), IPv6, mDNS, Windows networking, and miscellaneous other IP traffic. We then organize the categorized packets into contiguous *spans*; we consider outgoing packets only and ignore packets in-bound to the client (with the exception of deauthorization packets sent by the base-station, which terminate the current span).

How much time do clients spend performing mobility management tasks? Figure 8 presents a time series of the average fraction of time an active client spends in each type of span. The graph plots five-second bins, averaged over one-minute intervals. For

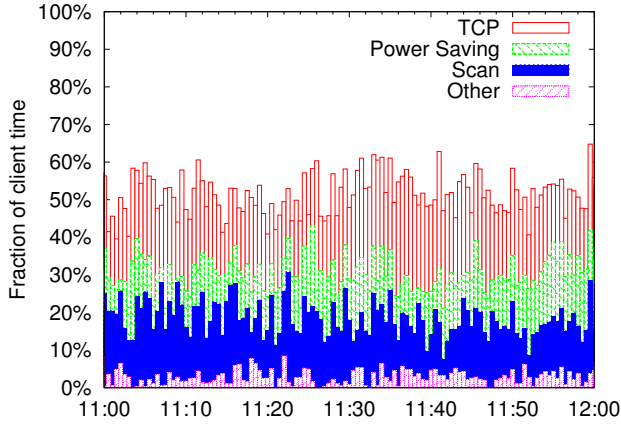


Figure 8: Time series of different types of spans.

clarity, we show only the categories that take the significant part of a client’s time: TCP, power save mode, and scanning; the “Other” category includes association, DHCP, DNS, ARP, and “misc” spans. Within each five-second bin, we calculate the fraction time each active client spends in each type of span, and normalize for the number of clients active in that bin. If a client sends no packets in a five-second interval, it is not counted. While the absolute fraction of active time in any interval depends on the bin size (clients are bursty; the longer the sample period the less dense the activity), the relative length of each type of span remains relatively constant. From the graph we can conclude that roughly one third of a clients active time is spent scanning or in some other maintenance activity (ARP, DHCP, association, etc.) — overhead directly due to mobility maintenance.

6.2 Impact of scanning

Figure 8 shows that 802.11 clients are constantly scanning for other APs that may offer better associations. If the station is otherwise idle at the time, scanning is inconsequential — at least from the point of view of the client. If the interface is busy, on the other hand, this behavior results in observable delay.

We therefore further refine our model so that we can quantify the delay observed by active 802.11 clients due to scanning. Our goal is to isolate those scan events that occur while the client was otherwise occupied. Because we do not know precisely what a given client is doing at any point in time, we have to make a conservative estimate. To do this, we label a TCP span “active” if the throughput is over 100 bps in a five-second bin. Otherwise we label it “idle”.

To what extent does mobility management interrupt client activity, thereby imposing undesirable delays? Going back to our trace, about 40% of stations have no interruptions at all, either because our criteria is too strict, the cards are smart enough to avoid interrupting, or the stations are just not active enough during our monitoring period. Figure 9 shows the CDF of interruption durations for the remaining 60% of the stations. The average interruption lasts for roughly 250 ms, and over 20% of interruptions last longer than one second. Most interruptions are caused by scanning behavior, but we also observe a substantial number of occasions where the station goes into power-save mode (i.e., sends a NULL packet with power save on, followed eventually by NULL packet with power save off). The “PSM Sleep” line in Figure 9 shows that, while PSM interruptions can be much shorter than scans, the average duration is roughly comparable and is unlikely to take longer than a second.

Short interruptions might be tolerable if they occurred infrequently,

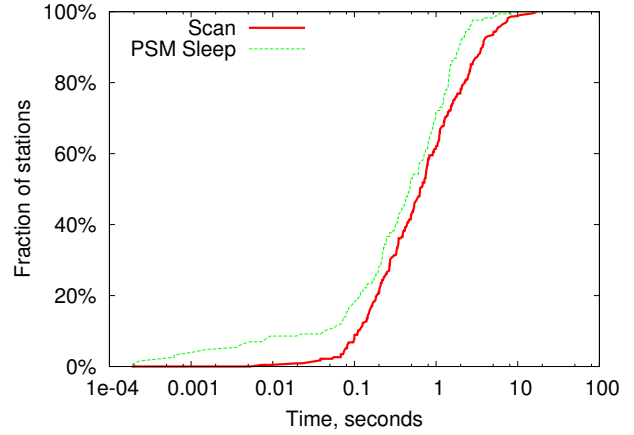


Figure 9: CDF of duration for scans and sleep periods.

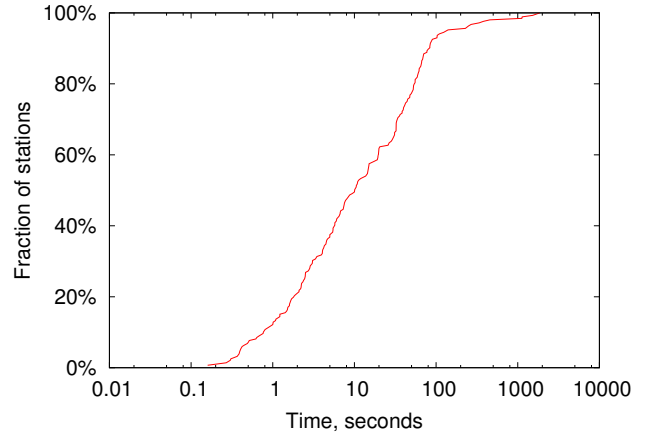


Figure 10: CDF of intervals between scans.

but Figure 10 shows that, for hosts that experience interruptions, they occur with wildly varying frequency. The average interrupted host is interrupted only once every 10 seconds or so. Such interruptions would not significantly affect the throughput, hence they fall in the “misc” category in our previous TCP analysis in Figure 7. However, 10% of the interrupted hosts are interrupted more than once a second, interruptions which are more likely to frustrate users using interactive applications like SSH.

In addition to delaying traffic at the scanning station itself, probes also tend to exacerbate the hidden terminal problem. Recall that the hidden terminal problem occurs when two stations transmit to the same third station simultaneously. A scan probe might be received by multiple nearby APs which are unable to hear each other. These “hidden” APs will attempt to respond simultaneously and may cause interference at the client. We are able to detect overlapping transmissions by comparing the start timestamp of every packet destined for an AP with the end timestamps of previous packets directed to the same AP. If they overlap, we mark both packets as having collided due to hidden terminals. We observe that over half of the stations sent probes that collided with another station’s packets, and, for the worst offenders, over 10% of their probes collided with other stations’ packets. As a result, hosts have to scan frequently to get responses from the available APs.

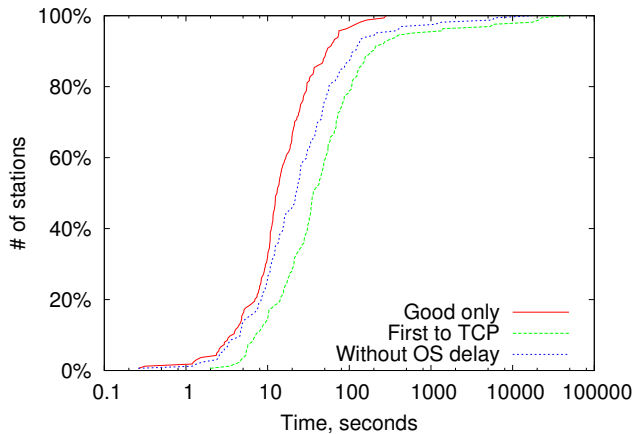


Figure 11: CDF of the delay experienced on startup by 802.11 clients in our network.

6.3 Startup

Next we describe how we model startup delays for when clients first connect to a wireless network. When a client first appears on the 802.11 network it must initiate a sequence of steps to effectively join the network before it can communicate at the IP level. The standard behavior of a host is as follows:

- Scan. Determine a candidate AP to associate with.
- Associate. Attempt to associate with the chosen AP.
- DHCP. Once the host has successfully joined the 802.11 network, it must obtain an IP address to begin communicating. In our environment, hosts obtain a dynamic IP address through DHCP.
- ARP. Equipped with an IP address, the first thing a host must do is determine the MAC address of the next-hop router to route IP packets towards their destination. Hence, the host will issue an ARP “who has” for the IP address of the next-hop router indicated by the DHCP server.
- DNS. Finally, once IP routing is established, the host can begin communicating with a non link-local IP address. Typically remote hosts are identified through domain names, so the host must resolve the name using the domain name service. Once DNS resolves the IP address of the destination, the host can begin sending actual data.

We begin by considering the delay associated with end-system startup. In an attempt to isolate those stations that are truly starting up—as opposed to simply re-associating after a period of idleness—we define a set of candidate selection rules. A station is deemed to be starting up if the first packet we see from it is a scan request. Because we are interested in the behavior of clients that should be able to use the network, we only consider stations that eventually succeed in associating with one of our access points and send at least one TCP packet.

How long are these delays? Figure 11 shows the distribution of startup times for those clients that do successfully connect to our network. There are three curves; “First to TCP” is the total wall-clock time from the first probe request to the first TCP segment. Surprisingly, most hosts take more than ten seconds before they begin communicating on the network, and the average host takes almost a minute. We conjecture, however, that the bulk of that time

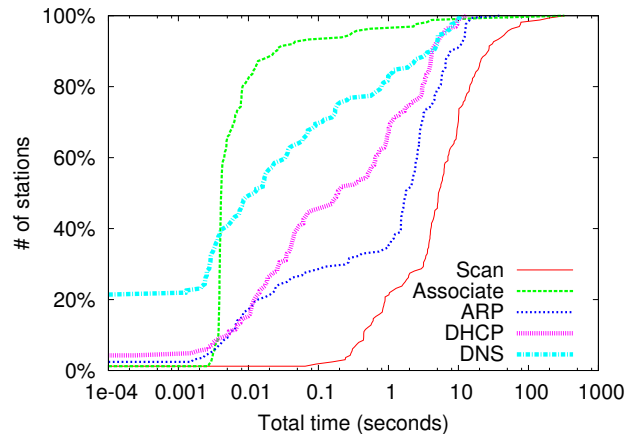


Figure 12: CDF of time spent in each successful phase of startup.

is spent idling—meaning the machine is not actively trying to make progress towards sending data.

To validate our conjecture, we attempt to determine if each successive span was successful or not—if successful, the time between spans is likely due to delays on the end host. In contrast, we assume that time between failed spans is due to some sort of network timeout. We define a scan to be successful if it is not followed by a subsequent scan; association, DHCP, and DNS are successful if the last packet in the span was outgoing from base station to client (i.e., an ACK). The “without OS delay” line removes estimated OS delays from the measured startup latency by subtracting idle time between successful spans under the presumption that any delay in initiating the subsequent span is due to the end host (i.e., the operating system has not yet initialized the network stack).

The average host spends almost eight seconds idling, presumably because the operating system is booting or resuming from power-save mode. Interestingly, however, if we sum only the duration of successful spans, we observe that the average host spends over 20 seconds during or after unsuccessful spans. The “good only” line represents a best-case scenario, with no idle time between stages. The question, then, is what’s going wrong—why the big gap between optimal and common case? To address this question, we first examine the successful spans.

Even the successful spans take a non-trivial amount of time. Figure 12 shows the breakdown of the various stages in the startup process. This breakdown uses span durations only, and ignores the time between spans. (Summing all curves from this graph together yields the “Good only” curve from above.) Clients spend the vast majority of this time scanning for an appropriate access point with which to associate. Association itself generally takes less than 10 ms, at which point communication with hosts beyond the access point can begin. DHCP, on the other hand, because it depends on a remote server, can take a variable amount of time. We will expand on the performance of DHCP in our environment in the next section. For now, however, we note it generally takes somewhere between 10 ms and five seconds to obtain an IP address.

Surprisingly, ARP, while frequently fast, takes longer than one second in more than half the cases. This delay results because most stations issue an “ARP to self”—an ARP “who has” request for their own IP address—to ensure no other station is using that IP address before they begin communication. By design, such an ARP request must timeout, hence the one-second delay. Note that some

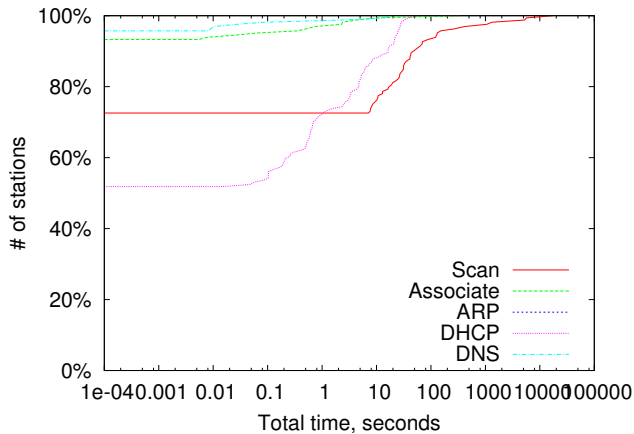


Figure 13: CDF of delays experienced by 802.11 clients due to timeouts.

DHCP Transactions	611 (100%)
Client had no known current lease	204 (33.39%)
Client had used 25% of current lease	288 (47.14%)
Client newly associated shortly before	193 (31.59%)
Client re-associated shortly before	56 (9.17%)
No valid reason determined	76 (12.44%)

Table 3: Potential reasons why clients initiated DHCP transactions over a day. For some DHCP transactions multiple potential reasons exist.

graphs start at greater than 0%; the clients not shown on the graph do not send those packets during startup. For example, over 20% of hosts do not issue a DNS query before starting a TCP connection, presumably because they are communicating with a manually specified IP address or because the corresponding DNS entry was previously cached.

Returning to unsuccessful spans, we observe that timeouts can be quite expensive. Figure 13 shows that while some stages, like DNS and association, frequently timeout in about 10 ms, they can take tens of seconds to complete in the worst case. The minimum DHCP timeout appears to be 100 ms and goes up from there. Failed scans are extremely expensive (a minimum of seven seconds) because a failed scan probably means there are no desirable access points in range. In this situation, it makes no sense to retry after short timeout, and most stations appear to wait for at least ten seconds before re-scanning the network. More interestingly, some hosts continue to scan for extremely long periods of time, presumably because they never find an AP they wish to join; i.e., they’re looking for a non-existent SSID.

6.4 Dynamic address assignment

Finally, we model dynamic address assignment using DHCP. DHCP is an inherent aspect of most 802.11 wireless networks. It is convenient for both users and network administrators, but the results above also indicate that DHCP can potentially impose noticeable and annoying delays to wireless users who desire and expect to be able to use the network quickly.

Clients initiate DHCP transactions for a variety of reasons: their last lease expired (or they had none), their existing lease is starting to expire (conservatively, when 75% of their current lease time remains), they associate with a new AP, or they re-associate with a previous AP. For instance, Table 3 shows a breakdown of the

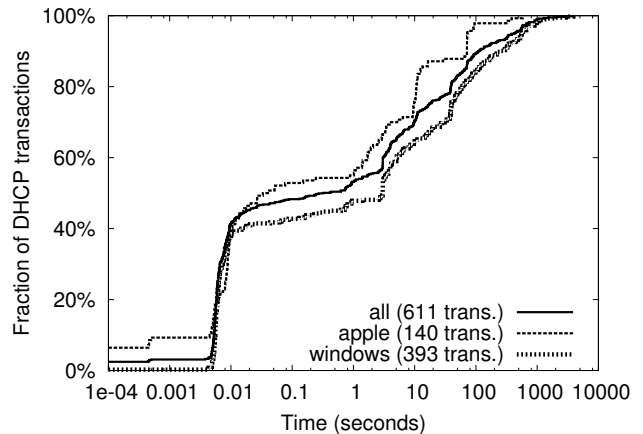


Figure 14: Distribution of DHCP transaction durations for an entire day.

reasons why clients in our building initiate DHCP transactions for an entire typical weekday of use. The dominant reason for DHCP transactions are clients contacting the server to start the lease renewal process. The vast majority of leases in our network are for three hours, so stable clients, once connected, initiate DHCP transactions throughout the day.

How long are DHCP transactions? The “all” line in Figure 14 plots the distribution of the duration of DHCP transactions for a typical weekday in the building. These results show that the majority of transactions complete in a reasonable amount of time: 75% of transactions complete in under six seconds. Users experiencing longer delays, however, are likely to be annoyed. On this day, over 10% of the DHCP transactions took longer than a minute to complete; for users connecting to the network for the first time that day, such a delay is quite noticeable.

Sometimes users wonder whether wireless behavior depends upon their operating system. Based on well-known Ethernet vendor codes for MAC addresses and the “Vendor Class” option in the DHCP protocol, we can determine the manufacturer of the operating system and networking hardware for almost all of the 186 stations in the trace. For comparison, we group the various versions of Microsoft Windows as “Windows” (118 stations) and hardware manufactured by Apple as “Apple” (51 stations) and show distributions for these groups as well. Apple clients consistently experience longer DHCP transactions than Windows clients. Apple hosts running OS X use the Zeroconf standard by default, which causes them to spend an additional ten seconds on startup. These clients optimistically attempt to renew their most recent lease (frequently from a private network at the user’s home), which is invalid in the campus building environment.

7. 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. As a first step in this direction, we have developed a set of models that take as input wireless trace data and can then accurately determine the impact of protocol behavior from the physical layer to the transport layer on transmissions in the trace. While some sources of delay can be directly

measured, many of the delay components, such as AP queuing, backoffs, 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 also explore an inherent class of overheads due to mobility management in 802.11 networks, including scanning for access points, association, ARP, DHCP, authentication, etc. To demonstrate the effectiveness of our models, we investigate the causes of transient performance problems from traces of wireless traffic in a four-story office building. 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.

Acknowledgments

We would like to thank a number of people for their contributions to this project. Lou Forbis dependably assisted us with all aspects of our wireless production network, and Jim Madden supported the operational needs of our network measurement efforts. We would also like to thank our shepherd Aditya Akella for his insightful feedback and support, and the anonymous reviewers for their valuable comments. Finally, Michelle Panik provided detailed feedback and copy-editing of earlier versions of this paper. This work was supported in part by the UCSD Center for Networked Systems (CNS), Ericsson, NSF CAREER grant CNS-0347949 and by U.C. Discovery CoRe grant 01-10099 as a Calit2-sponsored research project.

8. REFERENCES

- [1] P. Bahl, J. Padhye, L. Ravindranath, M. Singh, A. Wolman, and B. Zill. DAIR: A framework for managing enterprise wireless networks using desktop infrastructure. In *Proceedings of HotNets*, Nov. 2005.
- [2] A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan. Characterizing User Behavior and Network Performance in a Public Wireless LAN. In *Proceedings of ACM SIGMETRICS*, June 2002.
- [3] P. Barford and M. Crovella. Critical path analysis of TCP transactions. In *Proceedings of the ACM SIGCOMM Conference*, Stockholm, Sweden, Aug. 2000.
- [4] R. Chan, J. Padhye, A. Wolman, and B. Zill. A location-based management system for enterprise wireless LANs. In *Proceedings of NSDI*, Mar. 2007.
- [5] R. Chandra, V. Padmanabhan, and M. Zhang. Wifiprofiler: Cooperative diagnosis in wireless LANs. In *Proceedings of MobiSys*, June 2006.
- [6] 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.
- [7] Y.-C. Cheng, J. Bellardo, P. Benko, A. C. Snoeren, G. M. Voelker, and S. Savage. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. In *Proceedings of the ACM SIGCOMM Conference*, Pisa, Italy, Sept. 2006.
- [8] E. Daley. Enterprise LAN Grows Up, 2005. <http://www2.cio.com/analyst/report3401.html>.
- [9] K. N. Gopinath, P. Bhagwat, and K. Gopinath. An empirical analysis of heterogeneity in IEEE 802.11 MAC protocol implementations and its implications. In *Proceedings of WiNTECH*, 2006.
- [10] T. Henderson, D. Kotz, and I. Abyzov. The Changing Usage of a Mature Campus-wide Wireless Network. In *Proceedings of ACM Mobicom*, Sept. 2004.
- [11] IEEE Computer Society LAN MAN Standards Committee. IEEE Standard 802.11, Wireless LAN Media Access Control (MAC) and Physical Layer (PHY) Specifications, 1999.
- [12] A. P. Jardosh, K. N. Ramachandran, K. C. Almeroth, and E. M. Belding-Royer. Understanding Congestion in IEEE 802.11b Wireless Networks. In *Proceedings of ACM IMC*, Oct. 2005.
- [13] A. P. Jardosh, K. N. Ramachandran, K. C. Almeroth, and E. M. Belding-Royer. Understanding Link-Layer Behavior in Highly Congested IEEE 802.11b Wireless Networks. In *Proceedings of ACM E-WIND*, Aug. 2005.
- [14] J. Jun, P. Peddabachagari, and M. Sichitiu. Theoretical maximum throughput of IEEE 802.11 and its applications. In *Proceedings of IEEE International Symposium on Network Computing and Applications*, Apr. 2003.
- [15] A. Kochut, A. Vasana, A. U. Shankar, and A. Agrawala. Sniffing out the correct physical layer capture model in 802.11b. In *Proceedings of ICNP*, 2004.
- [16] D. Kotz and K. Essien. Analysis of a Campus-wide Wireless Network. In *Proceedings of ACM Mobicom*, Sept. 2002.
- [17] B.-J. Kwak, N.-O. Song, and L. E. Miller. Performance analysis of exponential backoff. *IEEE/ACM Transactions on Networking*, 13(2), Apr. 2005.
- [18] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Analyzing the MAC-level Behavior of Wireless Networks in the Wild. In *Proceedings of ACM SIGCOMM*, Sept. 2006.
- [19] A. Mishra, M. Shin, and W. Arbaugh. An Empirical Analysis of the IEEE 802.11 MAC Layer Handoff Process. *ACM Computer Communications Review*, 33(2), 2003.
- [20] 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*, Apr. 2000.
- [21] S. Rewaskar, J. Kaur, and F. D. Smith. A passive state-machine approach for accurate analysis of TCP out-of-sequence segments. *ACM Computer Communication Review*, 36(3), 2006.
- [22] A. Sheth, C. Doerr, D. Grunwald, R. Han, and D. Sicker. MOJO: A distributed physical layer anomaly detection system for 802.11 WLANs. In *Proceedings of MobiSys*, pages 191–204, June 2006.
- [23] J. Yeo, M. Youssef, and A. Agrawala. A Framework for Wireless LAN Monitoring and its Applications. In *Proceedings of ACM WiSe*, Oct. 2004.