Jigsaw: Solving the Puzzle of Enterprise 802.11 Analysis

Yu-Chung Cheng, John Bellardo, Péter Benkö[†]
Alex C. Snoeren, Geoffrey M. Voelker and Stefan Savage

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

ABSTRACT

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. In this paper, we present a system called Jigsaw that uses multiple monitors to provide a single unified view of all physical, link, network and transport-layer activity on an 802.11 network. To drive this analysis, we have deployed an infrastructure of over 150 radio monitors that simultaneously capture all 802.11b and 802.11g activity in a large university building (1M+ cubic feet). We describe the challenges posed by both the scale and ambiguity inherent in such an architecture, and explain the algorithms and inference techniques we developed to address them. Finally, using a 24-hour distributed trace containing more than 1.5 billion events, we use Jigsaw's global cross-layer viewpoint to isolate performance artifacts, both explicit, such as management inefficiencies, and implicit, such as co-channel interference. We believe this is the first analysis combining this scale and level of detail for a production 802.11 network.

Categories and Subject Descriptors
C.4 [Performance of Systems]: Measurement techniques
General Terms
Experimentation, Measurement, Performance
Keywords
Wireless networks, 802.11, measurement, monitoring

1. Introduction

In the last five years, 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 [3, 4, 5, 9]. However, the reality of these deployments can be quite complex. A large office building may have hundreds of wireless users interacting with dozens of access points under varying load and environmental conditions.

It is these interactions that make the dynamics of wireless network behavior so interesting, and yet so difficult to measure. Unlike their wired brethren, wireless networks are not well described as either a single broadcast channel nor as a graph of links. Whether any transmission is heard by a particular receiver is a function of many distinct factors including the ambient environmental interference, the sender's transmit power, the distance to the receiver, and the strength of any simultaneous transmissions on nearby channels heard by that same receiver. Further complicating this morass is the 802.11 media access control (MAC) protocol, which uses its own inferences about the physical layer to defer, schedule and retry transmissions. Finally, these networks are typically used to carry Internet traffic based on the TCP protocol that carries its own set of complex dynamics. It is no wonder that our understanding of these systems tends to be limited to either small controlled environments ("how much does interference between two radios impact throughput") or to large, but coarse measurements ("how long is the average TCP flow on a wireless network").

It is our belief that developing a deeper understanding of the dynamics and interactions in production wireless networks requires reconstructing their behavior in its entirety — measuring all frames and delivery outcomes across all wireless nodes. In the wired network this kind of measurement is typically achieved through passive monitoring, but in the wireless domain spatial diversity prevents any single monitor from capturing more than a small subset of traffic. Thus, extracting a global viewpoint requires many spatially dispersed monitors working in concert.

In this paper, we approach this problem from a systems point of view. We have developed a large-scale monitor infrastructure that overlays a building-scale production 802.11b/g network with over 150 passive radio monitors. These monitors in turn feed a centralized system, called Jigsaw, that uses this data to produce a precisely synchronized global picture of all physical, link-layer, network-layer and transport-layer activity. We believe our principal contributions are threefold:

- *Large-scale Synchronization.* We have designed and implemented a passive synchronization algorithm that can accurately synchronize over 150 simultaneous traces down to microsecond granularity. To accomplish this task at scale requires predicting the impacts of individual radio clock skews and leveraging frames overheard at multiple radios to opportunistically resynchronize.
- *Frame Unification.* We use this fine-grained synchronization to combine the contents of all traces, merging duplicates and

[†]Benkö is a visiting researcher at UCSD from the Traffic Analysis and Network Performance Laboratory (TrafficLab) at Ericsson Research, Budapest, Hungary.

SIGCOMM'06, September 11–15, 2006, Pisa, Italy.

constructing a synchronized single trace of all frame transmissions.

- *Multi-layer Reconstruction.* From raw frame data we reconstruct a complete description of all link and transport-layer conversations. To address the problem of missing data we have developed a set of inference techniques to deduce the presence, time placement, and even contents of missing data. Our analysis uses transport-layer information to resolve questions such as frame delivery, that can be inherently ambiguous at the link-layer alone.

The remainder of this paper is organized as follows: In Section 2 we review the important aspects of the 802.11 MAC protocol and prior work in wireless LAN measurement. In Section 3 we describe our measurement infrastructure and, in Section 4, we describe how we merge and synchronize passive traces. Section 5 then explains how we reconstruct link-layer and transport-layer viewpoints from raw frame data. Section 6 evaluates the coverage of the monitoring platform in the building. In Section 7 we demonstrate Jigsaw's capabilities through a set of measurements that exploit its global wireless network perspective. Finally, Section 8 summarizes our overall conclusions about constructing and using a large wireless monitoring infrastructure.

2. Background and related work

In this section we offer a brief tutorial in the operation of the 802.11 protocol, followed by a description of previous 802.11 measurement research.

The 802.11 media access control protocol is a CSMA/CA variant that uses "virtual carrier sense" to support an RTS/CTS 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. 802.11 also provides a link-layer retransmission facility. Thus when a station sends a unicast packet, the protocol requires the receiver to respond immediately with an ACK packet. 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). 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.

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.

Over the last decade, 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 [6, 7, 19] 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 [10, 11, 16, 18, 22, 23, 25, 26], industrial factories [24], corporate networks [2], and conference and professional meetings [1, 13, 14, 20, 21]). 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.

Recently, researchers have started addressing this question by extending wireless network measurement to passively capture and analyze link-level characteristics as well. Yeo *et al.* 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 [25, 26]. 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.* analyze the link-level behavior of traffic from a large IETF meeting using three monitors capturing traffic on orthogonal channels [13, 14]. 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 [17, 21]. 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.

Overall, our work substantially 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 [13, 21, 26], our measurement platform uses over 150 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 [13], do not merge traces among monitors [21], or merge only a small number of traces *offline* using globally observed events [17, 26]. 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.

3. Data collection

Any data analysis is ultimately predicated on the quantity, quality and precision of data that can be collected. While we believe

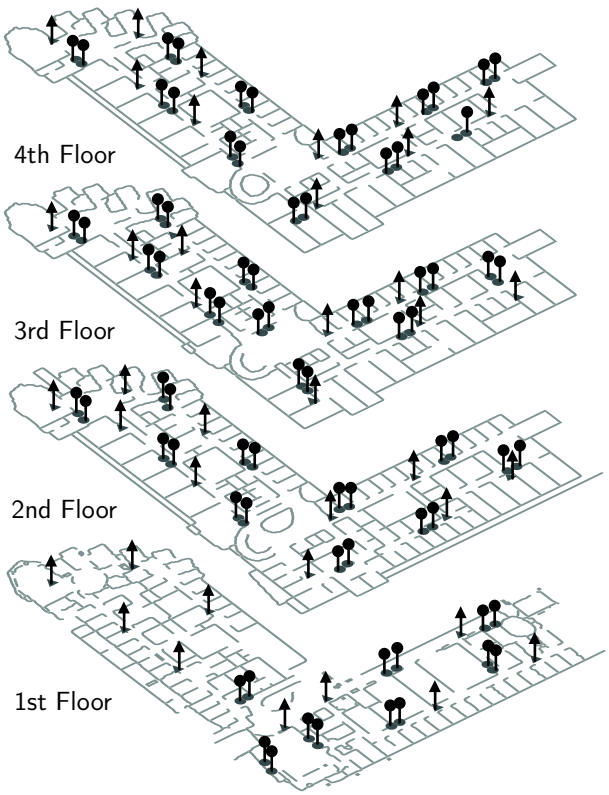


Figure 1: UCSD CSE building floorplan. This building comprises roughly 150,000 square feet over four floors (and a smaller basement, not shown). Circles indicate wireless sensor pods, and triangles indicate production access points.

that our analysis techniques are mostly generic, many of our design decisions *have* been informed by the capabilities of our infrastructure 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 156 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 infrastructure we have built to produce the raw traces for our analysis.

3.1 Environment

All of our measurement work takes place within the UCSD Computer Science and Engineering building, a large four-story structure shown in Figure 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.[1]

Between and among these production APs we have deployed a constellation of 39 wireless sensor pods (shown as pairs of circles).[2] Each pod in turn comprises 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. 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.

3.2 Hardware

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).[3]

Finally, trace data from all radios is sent via NFS to a single 2.8-GHz Pentium server hosting 2 GB of memory and 2 TB of storage (four 500-MB SATA disks in a RAID-0 configuration).

3.3 Software

Each monitor runs a version of Pebble Linux, using the *MadWifi* 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.

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 μs resolution clock to timestamp each packet as it is received. Our driver slaves this timestamp facility to the clock of a single radio, thereby recording frames at both radios using the same time reference.

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 64 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 metadata index record to facilitate subsequent accesses. Data and metadata are written to separate files via NFS, creating a new file pair each hour. In steady state, the NFS traffic across all 156 simultaneous feeds averages 2–10 MB/s.

4. Trace merging

Each individual trace 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. This merging procedure must satisfy three key requirements:

[1] In addition to the 39 production access points shown, the half-wing basement (not shown) houses five additional APs. We occasionally observed signals from 46 authorized APs from nearby buildings and 22 rogue APs (mostly outside the building).

[2] Our monitoring infrastructure does not cover the left wing of the first floor, which is not under our administrative control.

[3] Soekris Engineering uses an incompatible implementation of the 802.3af Power-Over-Ethernet standard and thus each system board is modified by hand to allow the HP switch to drive it.

1. *Unification*. Several radios may receive a particular frame which therefore appears in multiple traces. It is important that we identify these "duplicates" as corresponding to a single physical transmission. In some cases a received frame may not even be a perfect duplicate (*e.g.*, due to corruption or truncation) yet it should still be associated with the same transmission.

2. *Synchronization*. While the monitors timestamp each frame in each trace, the local clocks can vary significantly. 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 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" (20 μs 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.

Our approach, similar to Yeo *et al.*'s framework [25], 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 microsecond to cover 500 meters at 2.4 GHz. Consequently, we can treat the time at which a given frame is received by multiple monitors as a simultaneous event for all potential interactions. Thus, we can use frames heard by multiple monitors as a common reference point to synchronize the clocks at each monitor 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 frames with the same timestamps, thereby creating a single global trace. In the remainder of this section we describe Jigsaw'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 [8]. The two algorithms, however, diverge significantly in implementation due to the differing demands of their applications: Jigsaw must be 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 Jigsaw 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.

4.1 Bootstrap synchronization

Jigsaw 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 — albeit imaginary — coordinated time standard. We begin with the assumption that all local clocks run at the same rate, and then consider skew.

Let r_i denote the ith radio and let T_i represent the difference between its clock and "universal time" — the global time reference we hope to agree on. Let s_k denote the kth reference frame used to synchronize radios and let E_k be the set of pairs $< r_i, s_k >$ such that radio r_i receives frame s_k. Finally, let y_{ik} denote the local value of r_i's clock when it received s_k (defined if and only if $< r_i, s_k >$ is in E_k). Thus, when s_k has been received, the universal time can be defined as

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

To bootstrap synchronization, Jigsaw must find an assignment of T_i for each radio. Once the offset T_i is available, Jigsaw can place each frame s_k into universal time by adjusting its timestamp y_{ik}.

Ideally Jigsaw could locate a single 802.11 reference frame s_k where E_k contains every radio. Then y_{1k} could be picked arbitrarily to represent the initial universal time. Unfortunately, we cannot depend on such events in a large deployment since signal strength decays 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_is 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. Jigsaw examines the first second of data from each trace.[4] For each frame s_k in every monitor's trace, Jigsaw checks if it was also received by any other radios. If Jigsaw finds an identical frame heard by some radio r_i, it adds r_i into E_k. Note that 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, Jigsaw only uses "unique" frames for all synchronization activities. Generally, these are DATA frames that do not have the retransmit bit set.[5]

For every radio trace, Jigsaw picks the set E_k that contains the maximum number of radios and adds it into the synchronization set G. Jigsaw stops filling G when G contains an instance of each radio. Then, for each radio r_i, Jigsaw performs a breadth first search in G to reach r_1. Recently, Karp *et al.* [15] 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 μs). While 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 widened or more sets E_k added to G, but we have never had need to do this), Jigsaw is unlikely to find a path between radios on strongly disjoint channels. To fully synchronize across channels we exploit the fact that our monitors use a single local clock to timestamp frames received on both of their radios. 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] In this case, "the first second" refers to true time (UTC) as measured by the system clock on each monitor. Each monitor maintains their system clock within milliseconds using the NTP protocol and records this value in its traces. This is the only point at which the system clock time is ever used.

[5] Some Intel 802.11 implementations incorrectly retransmit data *without* the retransmit bit set, but thankfully this is rare.

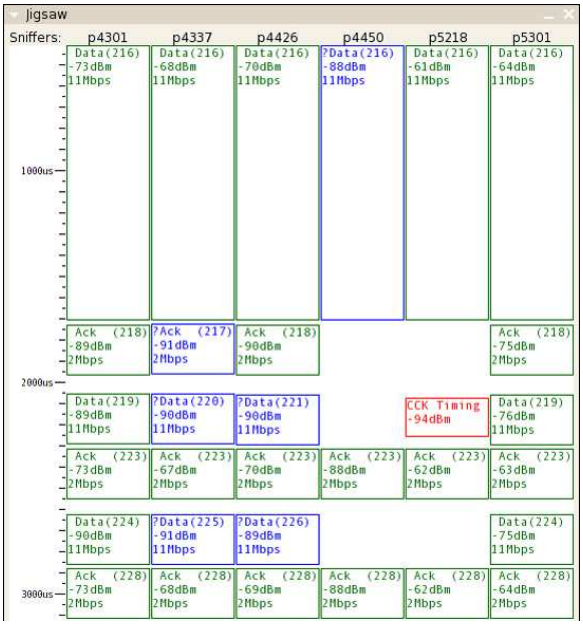


Figure 2: Jigsaw visualization of synchronized trace. Time appears on the x-axis in us and individual radios (only six shown here) on the y-axis. At roughly 400 us a client sends a data frame that is heard by all six radios. However, radio "p4450" is too far away (signal strength of -88 dBm); the data frame is corrupted and the subsequent ACK is not received. However, more than enough radios are present to construct a jframe for both parts of the frame exchange. At 2000 us a different client sends and it is heard by a different set of radios. Note that p5218 is too far away to even synchronize with the preamble.

4.2 Frame unification

After bootstrap synchronization, Jigsaw processes all traces in time order and unifies duplicate frames, called *instances*, into a single data structure called a *jframe*. Each jframe holds a (universal) timestamp, the full contents of the frame and the identity of the radios that heard each instance. Figure 2 provides an example of this source data as it is being unified. As part of the unification process, Jigsaw also aggressively resynchronizes the clocks between each trace to account for skew and drift. We describe the evolution of our algorithm below.

Basic unification

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

However, there are two problems with this approach. First, for large deployments the linear scan can have tremendous overhead. In our environment, most jframes contain 10 or fewer instances and yet we have over 150 simultaneous traces whose queues must be checked. To minimize this overhead, Jigsaw instead populates a single priority queue sorted by time with the earliest instance from each trace. To create a jframe, Jigsaw simply pops this queue until the timestamp of the next instance differs by a significant amount and groups the popped instances according to their content (it is still crucial to compare frame contents since it is possible that distinct frames may be transmitted simultaneously). Thus, the time to create a new jframe is linear in the transmission range of a particular frame, not the number of radios in the system.

The second problem is that each radio's clock skews over time. The 802.11 standard mandates an accuracy of at least 100 PPM (0.01%) and our experience is that Atheros hardware has far better frequency stability in practice. However, even good clocks eventually diverge. If the time offset between clocks becomes great enough, then some instances of a given frame may not be correctly merged into the same jframe. To mitigate this problem, we pop instances from the priority queue until the timestamp at the head of the queue exceeds some *time offset threshold* with respect to the candidate instances — *i.e.*, a "search window." Some of these additional frames may have identical content with the other candidates and Jigsaw will group them into the jframe, while the others are inserted back into the priority queue. Jigsaw uses the median instance timestamp as the universal timestamp for the resulting jframe.

Figure 3 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 3a shows Jigsaw 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 3b, unifies the frames into a jframe timestamped using the time offset of the median offset frame R_1.

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 Jigsaw unifies a set of frame instances the variance between their local instance timestamps and the jframe's universal 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 Jigsaw then uses to bring each of the associated traces back into synchronization. Figure 3b 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 tradeoff 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 timestamp for a frame instance — before resynchronizing. Figure 3a 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 μs. (Note that this does not limit the synchronization accuracy to 10 μs.)

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 peri-

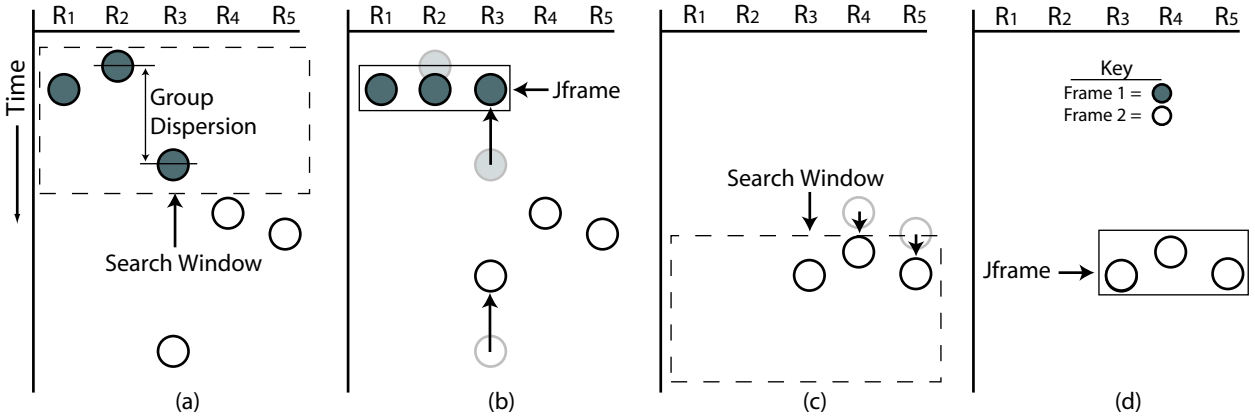


Figure 3: Unification and synchronization for two frame transmissions (dark and white) among five radios. Jigsaw (a) uses a search window to identify candidate frames for unification among radio queues; (b) unifies identical frames into a jframe timestamped with the median offset frame, and correspondingly adjust the time offsets of the other radio queues to account for skew; (c) adjusts offsets on other radio queues to account for their skew, and uses the search window for the next set of frames; (d) unifies the white frames into the next jframe.

ods (although rarely over 100 ms since this is roughly the period between AP beacon frames) 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 universal time will determine how quickly it will lose synchronization without readjustment. In practice, we have found that, with large numbers of radios, unless the search window is made dangerously large (100s of milliseconds) correct synchronization is lost quickly. However, many of these problems can be eliminated by incorporating measurements of per-radio clock skew into the synchronization algorithm. Thus, Jigsaw pro-actively adjusts the local timestamp of each instance to compensate for the clock skew on the radio receiving it. In addition, for large numbers of radios we have also found it important to compensate for clock *drift* — the change in skew over time — by using an exponentially weighted moving average of past skew measurements to predict future skew on a per-instance basis.

Figure 3c represents Jigsaw adjusting the skews of radios R_4 and R_5 by shifting the frames at the head of their queues. Jigsaw then repeats the unification process for the next set of frames in its search window, identifying the three white frames as identical. In Figure 3d, Jigsaw unifies them into a jframe timestamped with the median offset frame at R_5. For this jframe, however, the group dispersion is below the resynchronization threshold, and Jigsaw reduces overhead by skipping resynchronization for these frames.

Thus, Jigsaw can use almost every new data frame for continual resynchronization. This approach presents several key advantages compared to approaches that simply use reference beacons to synchronize [25]. 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

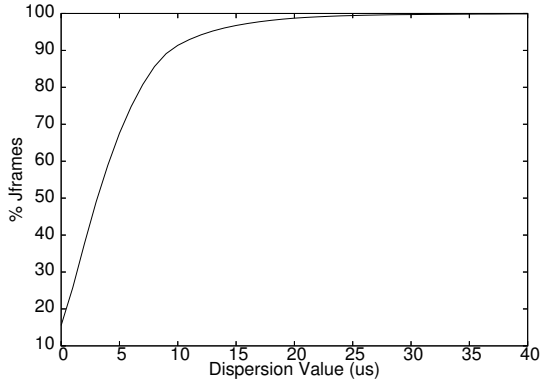


Figure 4: CDF of group dispersion across all frames.

in complexity. Jigsaw's synchronization and unification code totals roughly 4,000 lines of C++.

Figure 4 illustrates the current accuracy of our algorithm using a 10-ms search window. The graph shows the CDF of group dispersion values calculated for every jframe processed from 156 radios over a 24-hour period. For 90% percent of all jframes, the worst case time offset between any two radios is less than 10 μs, and 99% see a worst case offset under 20 μs. 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.

We emphasize Jigsaw's synchronization is designed for short-term 802.11 timing analysis. Jigsaw's calculated universal clock may diverge over time with respect to a true time standard. In fact, at the end of a day-long trace, Jigsaw's universal clock may be tens of seconds different than wall clock time. We are not concerned about minor drift between universal time and wall-clock time, however, because all of our analyses use universal time as the reference, so the calculations are internally consistent. Furthermore, such accuracy is sufficient for us to annotate the date and time in our results for diurnal interpretation.

5. Link and transport reconstruction

Having constructed a single global view of each observed physical event, the next task is to reconstruct each link-layer and transport-

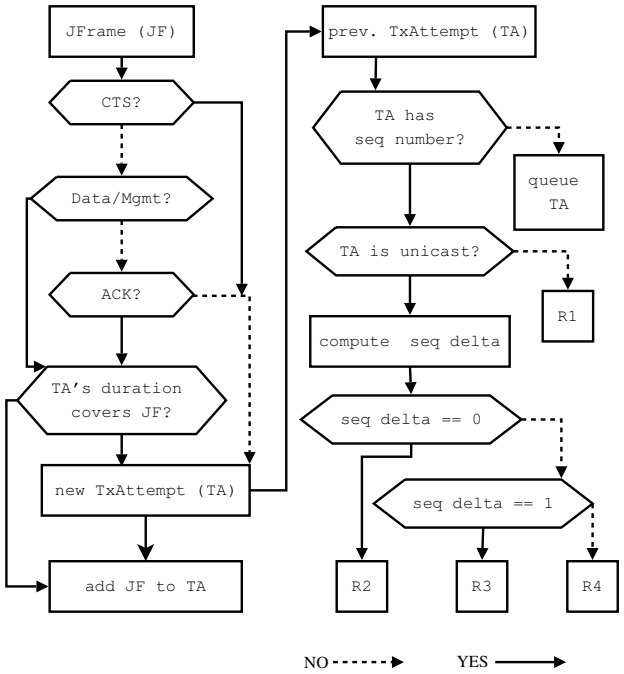


Figure 5: Simplified finite state machines used to assemble individual jframes into transmission attempts (left) and to compose transmission attempts into complete frame exchanges (right).

layer conversation in its entirety. In principle, this reconstruction is straightforward since Jigsaw provides a time-ordered list of all frames 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, Jigsaw must use inference to help reconstruct these higher-layer descriptions.

5.1 Link-layer inference

In reconstructing link-layer conversations, Jigsaw first identifies each *transmission attempt* from a sender (illustrated on the left side of Figure 5). 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 jframes 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 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. Space does not permit a complete description of all inference rules, but we sketch a simplified version of our state machine (shown on the right side of Figure 5). 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 that 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, 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. Unfortunately, the vantage point of a passive monitor does not allow this 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.

5.2 Transport inference

Our transport-layer analysis takes frame exchanges as input and reconstructs individual TCP flows based on the network and transport headers. We use a variant of Jaiswal *et al.*'s analysis (designed for wired passive monitors) to then infer connection characteristics (*e.g.*, RTT, RTO, fast retransmissions, segment losses, *etc.*) [12]. However, the passive wireless context has two ambiguities that differ from the wired environment. First, we may process frame exchanges in which it is unclear if the frame was actually delivered (as described previously). However, we can frequently use the transport-layer side effects of this frame 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 content, even with many different monitors, sometimes a frame exchange is completed but not observed *at all* by a monitor. However, if we observe a TCP acknowledgment that covers an 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.

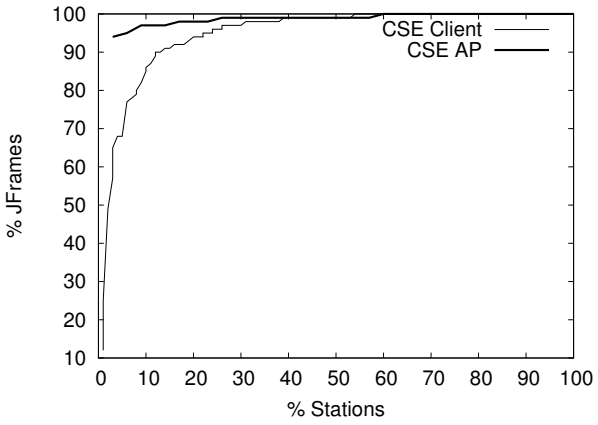


Figure 6: Coverage of frames transmitted by clients and APs.

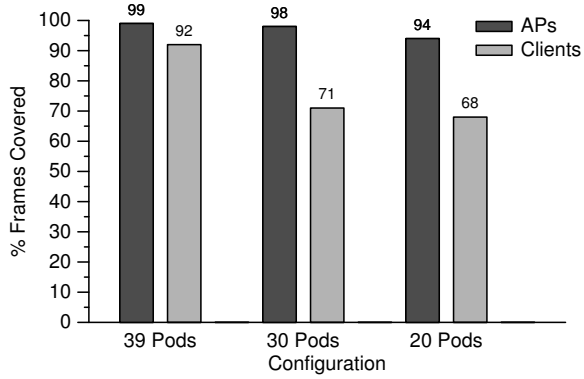


Figure 7: Coverage of frames transmitted by APs and clients for various configurations of sensor pods in the building.

6. 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.

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 is consistent with smaller-scale studies using similar wireless monitoring methodology: [13] reports a coverage of 80–97%, [21] reports 90%, and [26] 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 6 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 infrastructure—simply with fewer pods).

Figure 7 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 of all 39 pods (156 radios), 30 pods (120 radios), and 20 pods (80 radios); reducing to 10 pods creates partitions in 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 of client transmissions in our build-

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%)
Jframes	530 M
Jframe Events	1,580 M
Events/Jframe	2.97

Table 1: Summary of trace characteristics.

ing; in fact, we plan to improve client coverage further by adding additional pods where the monitoring platform has poor coverage.

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.

7. Analyses

In this section we perform a series of analyses on a trace of the building's wireless network captured by the monitor platform. We focus on preliminary analyses that exploit the global perspective afforded by distributed monitors. Our goal is not to be exhaustive, but rather to illustrate the unique capabilities of a global synchronized viewpoint. In future work, we intend to develop a comprehensive portfolio of analyses and use them to drive more complex 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, and then examine the effects of interference, the effects of 802.11g protection mode in networks with both 802.11b and 802.11g clients, and distinguishing link-layer and wired effects on TCP loss rate.

7.1 Trace summary

We start by summarizing the high-level characteristics of our trace and then show network activity over time. Table 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 events. Over 47% of these events 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. Jigsaw unifies 1.58 billion events (valid frames and a subset of associated error frames) into 530 million jframes, for an average of 2.97 events per jframe. In other words, on average the monitoring platform makes three observations of every observed transmission of a valid frame in the network.

Figure 8 shows network activity as a time series throughout the day at the granularity of one minute. Figure 8(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 8(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.

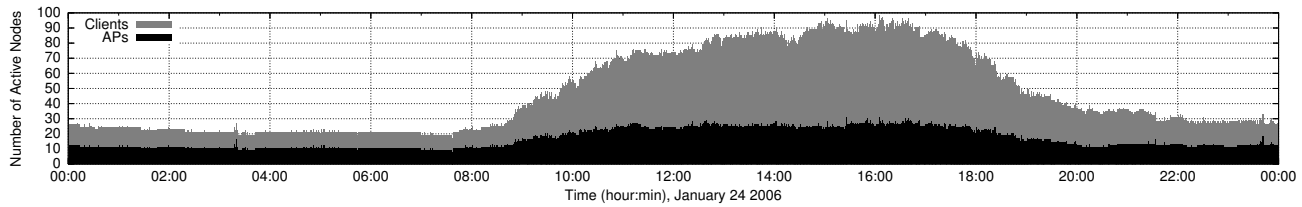
We also separate out two explicit categories of management traffic because of their high prevalence: "Beacon" shows the amount of periodic AP beacon traffic, and "ARP" shows the amount of ARP broadcast ARP traffic. Because APs broadcast beacon traffic independent of activity, beacon traffic is constant 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.[6] 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 by wired broadcasts, APs should add random jitter to the transmission time for broadcasts frames received from the wired network.

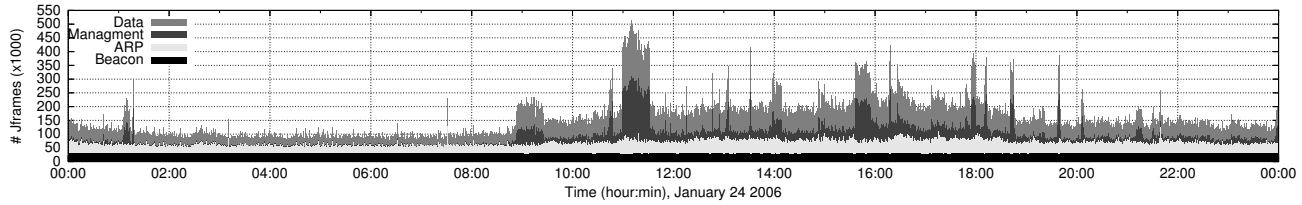
7.2 Interference

In this section, we analyze the extent of transmission interference experienced by nodes in our trace. Since the platform monitors orthogonal channels, adjacent-channel interference is rare and

[6] One 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.



(a) Active Clients and APs



(b) Traffic Breakdown

Figure 8: Time series of network activity throughout the day in one-minute intervals.

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 transmissions. 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:

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

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

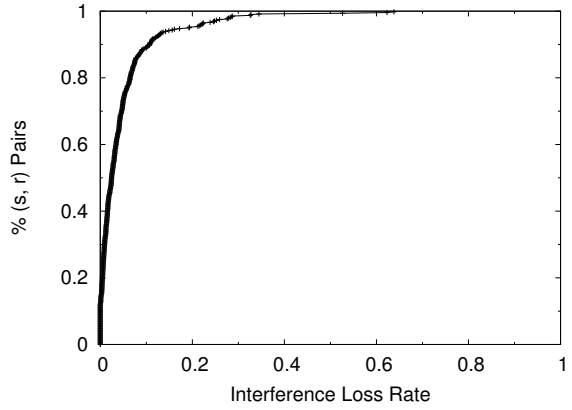


Figure 9: Interference loss rate X across (s, r) pairs.

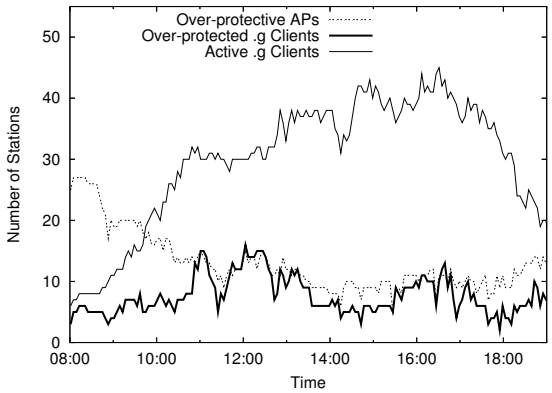


Figure 10: Overprotective APs and active 802.11g clients during the busy period of the trace.

this question, Figure 9 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 9 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.

7.3 802.11g protection mode

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 (Section 2). 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 10 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 a factor of two.[7] 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.

7.4 TCP loss rate inference

Using the TCP reconstruction algorithm described in Section 5, we assemble all flows that complete a handshake (eliminating port scans and connection failures). From these flows we then calculate the loss rate using a variant of Jaiswal *et al.*'s approach [12]. Then, by analyzing the frame exchanges making up each TCP segment we are able to determine if each loss — as seen by TCP — is due

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

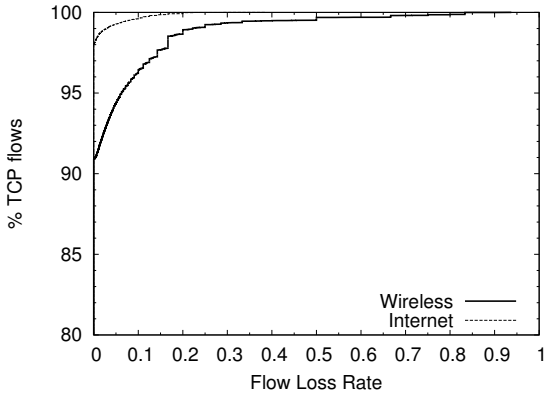


Figure 11: TCP loss rate.

to a lost 802.11 frame or some subsequent loss in the wired network. Figure 11 illustrates this data, showing — as expected — that the wireless component of TCP loss is dominant. What is important about this analysis is less the result itself than the capability to easily examine interactions between layers in our global trace.

8. Conclusion

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. To address this problem we have built a system called Jigsaw that unifies traces from multiple passive wireless monitors to reconstruct a global view of network activity in a production 802.11 network. We have described the algorithms used to scalably synchronize traces, unify common frames, and reconstruct the link- and transport-layer conversations embedded in those frames. To demonstrate our approach, we have deployed a large-scale instance of Jigsaw using over 150 monitors 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. Finally, for all its complexity, Jigsaw is only a building block. Our current work focuses on exploiting its capabilities to precisely answer the more salient questions that originally motivated our interest: "Why is the network slow?" and "How should it be fixed?".

For those interested in using or contributing to our efforts, the Jigsaw hardware specification and software are available for download at `http://wireless.ucsdsys.net`.

Acknowledgments

A number of individuals contributed to make this paper possible. Among them, Ryan Brown created the visualization of the UCSD CSE building, Greg Chesson of Atheros provided critical insight into the Atheros PHY implementation, Gordon Hamman arranged for all of our sensors to be wired and installed, Jim Madden supported the operational needs of our network measurement efforts and Bill Young helped us coordinate our technical activities within the department. We would also like to thank John Zahorjan for his insightful feedback, 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), the Sloan Foundation, Ericsson, NSF CAREER grant CNS-0347949 and by U.C. Discovery CoRe grant 01-10099 as a Calit2-sponsored research project.

9. REFERENCES

[1] 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*, 2002.

[2] M. Balazinska and P. Castro. Characterizing Mobility and Network Usage in a Corporate Wireless Local-Area Network. In *Proceedings of USENIX MobiSys*, 2003.

[3] Campus Computing. 2005 National Survey of Information Technology in U.S. Higher Education, 2005.

[4] E. Daley. Enterprise LAN Grows Up, 2005. `http://www2.cio.com/analyst/report3401.html`.

[5] Dell'Oro Group. Wireless LAN Five Year Forecast Report, 2006.

[6] D. Duchamp and N. F. Reynolds. Measured Performance of a Wireless LAN. In *Proceedings of IEEE LCN Conference*, 1992.

[7] D. Eckardt and P. Steenkiste. Measurement and Analysis of the Error Characteristics of an In-Building Wireless Network. In *Proceedings of ACM SIGCOMM*, 1996.

[8] J. Elson, L. Girod, and D. Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *Proceedings of OSDI*, 2002.

[9] Gartner. Market Share: Wireless LAN Equipment Worldwide, 2005.

[10] T. Henderson, D. Kotz, and I. Abyzov. The Changing Usage of a Mature Campus-wide Wireless Network. In *Proceedings of ACM Mobicom*, 2004.

[11] F. Hernández-Campos and M. Papadopouli. A Comparative Measurement Study of the Workload of Wireless Access Points in Campus Networks. In *Proceedings of IEEE PIMRC*, 2005.

[12] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP Connection Characteristics from Passive Measurements. In *Proceedings of IEEE Infocom*, 2004.

[13] 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*, 2005.

[14] 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*, 2005.

[15] R. Karp, J. Elson, D. Estrin, and S. Shenker. Optimal and Global Time Synchronization in Sensornets. Technical Report CENS-TR0012, CENS, UCLA, 2003.

[16] D. Kotz and K. Essien. Analysis of a Campus-wide Wireless Network. In *Proceedings of ACM Mobicom*, 2002.

[17] 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*, 2006.

[18] M. McNett and G. M. Voelker. Access and Mobility of Wireless PDA Users. *Mobile Computing and Communications Review*, 9(2), 2005.

[19] G. T. Nguyen, R. H. Katz, B. Noble, and M. Satyanarayanan. A Trace-Based Approach For Modeling Wireless Channel Behavior. In *Winter Simulation Conference*, 1996.

[20] K. N. Ramachandran, E. M. Belding-Royer, and K. C. Almeroth. DAMON: A Distributed Architecture for Monitoring Multi-hop Mobile Networks. In *Proceedings of IEEE SECON*, 2004.

[21] M. Rodrig, C. Reis, R. Mahajan, D. Wetherall, and J. Zahorjan. Measurement-based Characterization of 802.11 in a Hotspot Setting. In *Proceedings of ACM E-WIND*, 2005.

[22] D. Schwab and R. Bunt. Characterising the Use of a Campus Wireless Network. In *Proceedings of IEEE Infocom*, 2004.

[23] D. Tang and M. Baker. Analysis of a Local-Area Wireless Network. In *Proceedings of ACM Mobicom*, 2000.

[24] A. Willig, M. Kubisch, C. Hoene, and A. 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.

[25] J. Yeo, M. Youssef, and A. Agrawala. A Framework for Wireless LAN Monitoring and its Applications. In *Proceedings of ACM WiSe*, 2004.

[26] J. Yeo, M. Youssef, T. Henderson, and A. Agrawala. An Accurate Technique for Measuring the Wireless Side of Wireless Networks. In *Proceedings of USENIX/ACM WiTMeMo*, 2005.