Analyzing the MAC-level Behavior of Wireless Networks in the Wild

Ratul Mahajan
Microsoft Research

Maya Rodrig
University of Washington

David Wetherall
University of Washington

John Zahorjan
University of Washington

ABSTRACT

We present Wit, a non-intrusive tool that builds on passive monitoring to analyze the detailed MAC-level behavior of operational wireless networks. Wit uses three processing steps to construct an enhanced trace of system activity. First, a robust merging procedure combines the necessarily incomplete views from multiple, independent monitors into a single, more complete trace of wireless activity. Next, a novel inference engine based on formal language methods reconstructs packets that were not captured by any monitor and determines whether each packet was received by its destination. Finally, Wit derives network performance measures from this enhanced trace; we show how to estimate the number of stations competing for the medium. We assess Wit with a mix of real traces and simulation tests. We find that merging and inference both significantly enhance the originally captured trace. We apply Wit to multi-monitor traces from a live network to show how it facilitates 802.11 MAC analyses that would otherwise be difficult or rely on less accurate heuristics.

Categories and Subject Descriptors

C.4 [Performance of systems]: Measurement techniques

General Terms

Measurement, performance

Keywords

Passive monitoring, measurement tool, 802.11 MAC

1. INTRODUCTION

Measurement-driven analysis of live networks is critical to understanding and improving their operation (e.g., [3, 28, 13, 18, 22, 24]). In the case of wireless, however, very little detailed information is currently available on the performance of real deployments, even though wireless protocols are a subject of intense research and standardization [1, 15, 20, 16, 25]. This is particularly surprising given the abundance of live networks, such as public hotspots, and the apparent ease with which they can be measured.

We believe that this paradoxical state of affairs is due to fundamental challenges in measuring and analyzing live wireless networks. For instance, consider the task of determining how often clients retransmit their packets. This is one basic indicator of wireless performance that we may wish to measure. Most studies of operational wireless networks to date are based on SNMP logs from the APs or packet traces from the wire adjacent to the APs [4, 5, 7, 8, 23, 26]. However, neither method is sufficient for our task. Traces from wired segments omit information about wireless retransmissions altogether. Similarly, AP logs provide coarse information on transmissions and receptions of the AP but not on those of the clients. It is not simply a matter of granularity. Even a complete packet trace from the wireless side of the AP is not sufficient as it does not reveal client transmissions that the AP did not receive.

A potential solution to this problem is to instrument the entire network to obtain traces of activity at all the clients and the APs. This approach has been successful in testbed settings [2, 10, 29]. However, it is impractical for widespread use in operational networks that may have many heterogeneous and transient clients belonging to different users.

The remaining approach, which we explore in this paper, is to depend on traces obtained via passive monitoring. Here, one or more nodes in the vicinity of the wireless network record the attributes of all transmissions that they observe. This approach has the large practical benefit that it is almost trivial to deploy. Unfortunately, traces collected using it are limited in several respects. First, they will necessarily be incomplete, due to packet drops caused by weak signals and collisions. It is not even easy to estimate how much information is missing, which renders the trace of unknown quality. Second, the traces do not record whether the packets were successfully received by their destinations. That information is needed to compute many fundamental performance measures such as reception probability and throughput. Third, traces only record information about packet events and omit other important network characteristics, such as the offered load on the network at any given instant. It is challenging to estimate these characteristics because, unlike instrumentation, passive monitoring lacks access to the internal state of the nodes. These problems are also present in the wired domain [17], but the magnitude of their effects in the wireless domain makes them qualitatively different and more challenging.

Our goal is to tackle these challenges and develop sound methodologies for analyzing traces collected via passive monitoring of wireless networks. This lets us apply passive monitoring to live networks and investigate questions that are not easy to answer today. We are particularly interested in moving beyond basic performance measures (e.g., how often do clients retransmit their packets?) to higher-level questions concerning the network (e.g., what is the average asymmetry in the loss ratio between two nodes?) and its interaction with the protocol (e.g., how does network performance vary with offered load?).

In this paper, we take our first steps toward this goal and present the design and implementation of a tool called Wit. Wit is composed of three components that tackle each of the challenges that we iden-

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

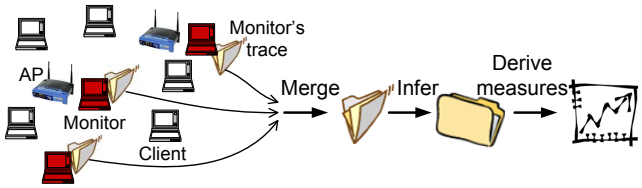


Figure 1: ***The pipeline of our passive monitoring approach.***

tify above to reconstruct a detailed record of system-wide behavior. To help improve the inherently incomplete view of a single monitor, the first component merges the independent views of multiple monitors into a single, consistent view. The second component uses novel inference procedures based on a formal language model of protocol interactions to determine whether each packet was received by its destination. It also infers and adds to the trace packets that are missing from the merged trace. The third component derives network-level measures from the enhanced trace. In addition to simple measures such as packet reception probabilities, it estimates the number of nodes contending for the medium as a measure of offered load. This characterization has not been previously achieved to our knowledge, and we expect that future research will add more techniques to our final component.

To evaluate the accuracy and completeness of the trace reconstructed by Wit, we use a mix of real traces and simulation tests. We find that the independent views of multiple monitors can be precisely merged, which provides an effective way to gain coverage. Inference correctly determines the reception status of packets in the vast majority of cases and is effective at adding packets that are not recorded. It also supports estimates of trace completeness. We find that Wit can accurately derive network-level measures, such as the number of contenders, from the enhanced trace.

To further demonstrate its abilities, we apply Wit to multi-monitor traces that we collected at the SIGCOMM 2004 conference. We show how Wit supports the straightforward computation of metrics, such as packet reception probabilities, in a manner that is more accurate than existing heuristics. We uncover MAC-layer characteristics of this environment that, to our knowledge, cannot be obtained by other methods. For instance, we find that the network was dominated by periods of low contention during which the medium was poorly utilized even though stations were waiting to transmit packets. Our analysis suggests that the 802.11 MAC is tuned for high contention levels that are uncommon for our measured network and motivates the need for more adaptive MACs.

2. OUR PASSIVE MONITORING APPROACH

To make passive monitoring effective, we develop a three-phase approach to address each key challenge that we identified: i) a single monitor will miss many transmitted packets; ii) monitors do not log whether a packet was received by its destination; and iii) monitors do not log network-level information. Figure 1 shows the resulting pipeline. The first phase uses multiple monitors to capture more wireless activity and merges the independent views into a single, more complete rendition. It must precisely time synchronize the individual traces and identify and remove duplicate packets.

The second phase infers which packets were received by their destinations and also infers packets that were not logged by any monitor – even a dense array of monitors will miss many packets. This phase uses a formal language technique based on the observation that, for many packets, the subsequent packet exchanges of the 802.11 protocol reveals both pieces of information.

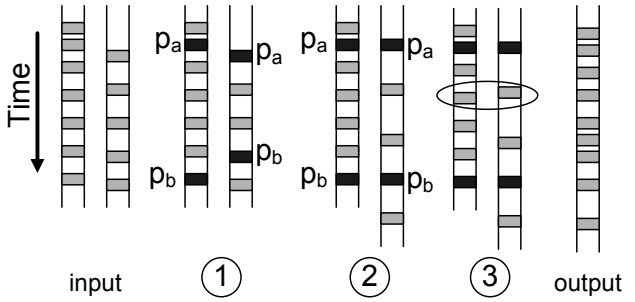


Figure 2: ***Our procedure to merge two traces. The ladders represent traces, and the shaded boxes represent packets. 1. Identify common references. 2. Translate timestamps of the second trace. 3. Identify and remove duplicates.***

The final phase derives network-level measures from the now enhanced packet trace. In contrast to the first two, this phase has an open-ended goal of deriving all measures that are needed to answer the questions of interest. Some measures, such as packet reception probability and throughput, are straightforward to compute, while others require new techniques. We present a novel technique to estimate the number of stations contending for the medium. Future derivations might estimate, for instance, which nodes defer to one another and detect the presence of hidden terminals.

We describe each phase in more detail below. Their implementation in Wit is described in the following section.

2.1 Merging

The input to the merging process is a number of packet traces, each collected by a different monitor. In each, the timestamp on a packet reflects a monitor's local view of when the packet was received. The goal of merging is to produce a single, consistent timeline for all the packets observed across all the monitors, that is, to eliminate duplicates and to assign coherent timestamps to all packets regardless of the monitor that captured them.

To be useful, the merged trace must have timestamps accurate to a few microseconds, the granularity of 802.11 MAC-level operations. The challenge is that the monitors' clocks are not synchronized at this granularity, and in fact may have significantly different skews and drifts. Eliminating duplicates is another challenge. Only a few types of 802.11 packets carry information that is guaranteed to be unique over even a few milliseconds, and distinct transmissions of bit-for-bit identical packets can happen on timescales of a few hundred microseconds. The only way to distinguish duplicates across traces from distinct transmissions is by time. This stresses the need for accurate timestamps in the merged trace.

We extend the scheme of Yeo *et al.* [27] to accurately merge realistic datasets with many long traces in a manner that is robust to the vagaries of data from live networks. Like Yeo *et al.*, we leverage "reference" packets that can be reliably identified as being identical across monitors. We first consider how to merge a pair of traces and then extend merging to an arbitrary number.

We use the three step process shown in Figure 2 to merge pairs of traces:

1. Identify the reference packets common to both monitors. We use beacons generated by APs as references, since they carry a unique source MAC address and the 64-bit value of a local, microsecond resolution timer. This is sufficient in practice, though other types of packets could be used if necessary.

2. Use the timestamps of the references to translate the time coordinates of the second monitor into those of the first, as shown in the transition from Step 1 to 2 in Figure 2. We independently translate each interval spanned by successive pairs of references with a simple linear function: the timestamp interval in the second trace is "stretched" or "shrunk" to match the first trace, and then a constant is added to align the two. The resizing reflects relative clock drift and the alignment reflects relative skew.

3. Remove duplicates by identifying them as packets of the same type, with the same source and destination and with timestamp difference less than half of the minimum time to transmit a packet (106 μsecs for 802.11b). This ensures that distinct packets from the same source are never mistakenly identified as duplicates.

Linear timestamp translation in Step 2 assumes that the relative clock drift is constant. If that were true over long intervals, it would suffice to merge based on only two references. However, we find that even our most reliable time sources exhibit varying relative drift. This is why we use multiple references and interpolate independently between successive pairs, reducing the impact of clock fluctuations to short intervals. For this to be effective, common references must be found with reasonable frequency, which we quantify later. In contrast, Yeo *et al.* use linear regression over the references in the entire trace to translate the time coordinates of the second monitor [27]. We find in Section 4.2 that the precision of this method degrades over long traces because of non-linearities in clock behavior.[1]

In theory, the two-trace procedure could be directly extended to merge arbitrary numbers by translating the time coordinates of all the monitors into those of the first. Yeo *et al.* adopt this approach. In practice, however, we find that the scarcity of common references is a limiting factor when monitoring a large network – it is unlikely that any one monitor will have enough references in common with each of the others. To circumvent this problem, we use a waterfall merging process: we merge the traces from two monitors, then add the trace from a third monitor to the merged trace, and so on. This approach has a somewhat longer running time.[2] The benefit is improved precision: as each additional trace is merged, it introduces new references into the partially merged result, making it easier to find a set of references shared with the next trace dense enough to overcome variable clock drift.

2.2 Inferring Missing Information

The inference phase serves two purposes. First, it uses the information in packets that the monitors did capture to infer at least some that they did not. It then synthesizes as complete as possible versions of such packets, improving the effective capture capability. In this role, inference is similar to using more monitors and then merging their views. Second, it introduces an entirely new class of information to passive traces – an estimate of whether packets were received by their destination.

The key insight behind our inference technique is that the packets a node transmits often imply useful information about the packets it must have received. For instance, an AP sends an ASSOCIATION RESPONSE only if it recently received an ASSOCIATION REQUEST. If the trace contains a response, but no matching request, we know that the request was sent and that it was successfully received. We also know that the sender of the request was the destination of the response. We can thus reconstruct many key attributes of the missing packet. While others have used such heuristics [6, 11, 21], we systematically automate this kind of reasoning.

2.2.1 The Formal Language Approach

We cast the inference problem as a language recognition task. Sentences in the language represent legal sequences of packets exchanged by two endpoints that follow the protocol. We call these packet exchanges *conversations* and define them at the granularity of logical 802.11 operations (e.g., all packets involved in an association attempt, or an exchange involving RTS, CTS, DATA, and ACKs to successfully convey a single data packet). Although longer conversations can be defined (e.g., association must precede data transmission) to enable a slightly larger set of inferences, the practical benefit of doing so is tiny; the additional inferences are about relatively rare events.

We view the input trace as interleaved partial sentences from the language. The interleaving stems from overlapping conversations between distinct endpoint pairs. A similar view of packet traces is taken in the context of passive testing of protocol implementations [14]. Our goal is different, however: to find valid sentences in the language that account for what is observed in the input trace. Thus, we do not simply ask "Is this sentence in the language?" Rather, we presume that there was a sentence in the language for which we see only some of the symbols and ask what complete sentence it was likely to have been.

We use regular languages as our choice of the formal language because they are recognizable by finite state machines (FSMs) which have efficient implementations. FSMs also afford an efficient way to extend traditional language recognition in a way that allows sentence reconstruction from partial information, as described below. Our short conversations can be easily described using FSMs.

2.2.2 Processing the Trace

Assume that the FSM (and so the language) for our protocol has been defined. To infer missing information using it, we scan the trace and process each packet as follows:

1. Classify We map packets to symbols of the language based primarily on their type. We also use the values of the retry bit and the fragment number field in forming symbols, which provides some additional leverage in making inferences, at the cost of a somewhat larger symbol set and FSM. Additionally, we identify the conversation of the packet based on its source and destination. For packets without the source field (ACKs and CTSs), we deduce the source from earlier packets. Non-unicast packets are considered conversations of a single packet.

2. Generate Marker Our language contains an artificial symbol, which we call the *marker*. We introduce a marker if the currently scanned packet indicates that an ongoing conversation has ended. This occurs under one of the following conditions. First, the sequence number field signals a new conversation between the endpoints. Second, for non-AP nodes, the other endpoint of the current packet is different from the earlier one; only APs can have multiple simultaneous conversations. Third, there is no legal transition in the FSM for the current symbol; if nodes correctly implement the 802.11 protocol, our FSM construction (described below) ensures that there is always a transition for packets in the current conversation. Fourth, a timeout interval has passed since the last seen activity for the conversation.

3. Take FSM Step If a marker was generated, first take a step in the FSM based on the marker. By construction, this causes

[1] Recent communication with Yeo informed us that they have now evolved a technique that is similar to ours.

[2] It takes $O(N^2Z)$ time, where N is the number of traces and Z the number of packets in each. Simultaneous merging takes $O(N \log(N)Z)$ time.

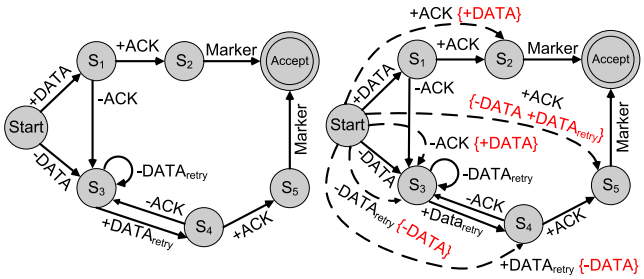


Figure 3: Left: ***An FSM for our simplified example. A '+' indicates that the packet was received by its destination, and a '−' indicates that the packet was lost.*** **Right:** ***The same FSM after augmentation of the Start state (only). The dashed edges are the augmented ones and the symbols in braces are their annotations.***

a transition to the accept state, closing the current conversation and placing the FSM in the start state. The path taken from the start to the accept state reveals information missing from the trace, as explained shortly. Now take a step in the FSM based on the symbol for the current packet.

While the first two steps involve some 802.11-specific decisions, the third step is entirely independent of the protocol being analyzed.

Key to this process is the construction of the FSM, which requires elaboration. We cannot simply use an FSM corresponding to the protocol because packets (i.e., sentence symbols) are missing from the trace and because we want to use the FSM to estimate which packets were received by their destination. We extend traditional FSM matching to address these issues.

We explain our method in the context of a simplified version of 802.11 data exchange conversations in which there are no fragments, and instead of quitting after a configured number of attempts, nodes retransmit the data packet until they receive an ACK. An FSM for this simplified version is shown on the left in Figure 3. (In contrast to Figure 3, our production FSM for the complete 802.11 protocol has 339 states.)

Inferring Packet Reception We first explain how we infer packet reception, assuming for now that all transmitted packets are present in the trace. To infer packet reception, we nondeterministically walk both the packet received and packet lost edges and encode the current state as the distinct set of *paths* traversed so far. When the accept state is reached, the edges traversed on the paths from the start state can be examined to determine whether each packet was received. (We describe shortly how we handle multiple paths to the accept state.)

For instance, as shown on the left in Figure 3, if the first packet seen is a DATA packet, we transition along both the +DATA and −DATA edges, producing the state $\{Start{\rightarrow}S1,\ Start{\rightarrow}S3\}$. If an ACK is seen next, the FSM state after the transition becomes $\{Start{\rightarrow}S1{\rightarrow}S2,\ Start{\rightarrow}S1{\rightarrow}S3\}$. In this case, the original ambiguity about the reception of the DATA packet has been resolved: it was received since both live paths traverse the +DATA edge. In contrast, if the second seen packet is a DATA$_{retry}$, the FSM state will be $\{Start{\rightarrow}S3{\rightarrow}S4\}$, and we can infer that the DATA packet was lost.

Inferring Missing Packets We now explain how we infer packets that are missing from the trace. With missing packets, there may be no legal transition for the current symbol. For instance, in our simplified example, if the first packet encountered for a conversation is a DATA$_{retry}$, there is no legal transition out of the start state. Our solution is to augment the FSM with additional edges. Abstractly, for each pair of states $(S_i,S_j) \neq (Start, Accept)$, we add an edge from S_i to S_j for each distinct trail (or, a path with no repeated edges) from S_i to S_j, labeling it with the final symbol of the trail. We annotate each augmented edge with the traversed trail's prefix, i.e., the path without the final symbol. The annotation represents packets that must be missing if the edge is traversed to reach the accept state. Example augmented edges can be seen on the right side of Figure 3. For instance, the edge between $Start$ and S_4 can be taken upon observing a DATA$_{retry}$ and its annotation indicates a missing DATA packet (which was lost).

We move non-deterministically in the augmented FSM until the accept state is reached. At this point, there may be multiple paths from $Start$ to $Accept$, all of them consistent with the captured packets. To select, we assign weights to paths and select the lowest weight one.[3] The weight of a path reflects the number of packets that it indicates as missing and the rarity of those packets types. Specifically, it is the sum of the weights of its edges. Unaugmented edges, which correspond to captured packets, have zero weight. The weight of an augmented edge is the sum of the weights of the symbols in the annotation. Symbol weights are inversely proportional to their frequencies in the trace. (We find that our inferences are similar even when we use the logs of these values, which translates the decision to the minimum product of the inverse frequencies, rather than their sum.) This weighting method prefers the shorter of two paths when the symbols of one are a subset of the other, thus producing conservative estimates of missing packets.

When the path weight function is a linear operator, as in our case, a straightforward optimization simplifies FSM construction, without impacting results. If there are multiple trails from S_i to S_j ending with the same symbol, only the lowest weight one needs to be considered.[4] The right side of Figure 3 shows the FSM for our example after the $Start$ state (only) has been augmented using this optimization.

As a final step when the accept state is reached, we synthesize any missing packets along the selected path. We cannot always infer the exact properties of a missing packet but can often do so. Properties that are relevant for MAC-level analysis include packet size and transmission time and rate, and which of these we can infer depends on the details of the 802.11 protocol. The size of certain packet types, such as ACK, RTS and CTS, is fixed. For others, such as DATA packets, the size can be inferred if a retransmission of the packet is observed. The transmission time of a missing packet can be inferred if there exists a captured packet relative to which it has a fixed spacing; for instance, the transmission time of a DATA packet can be inferred from that of the corresponding ACK. The transmission rate of certain packet types, such as PROBE REQUEST, is usually fixed for a client, and for certain other types, such as ACK, it depends on the rate of the previous, incoming packet (i.e., DATA). However, the rate of missing DATA packets cannot be inferred unless the rate adaptation behavior of the sender is known.

[3]While we pick a single final path, our framework allows us to enumerate others, along with measures of their plausibility. Also, as a practical matter, our weighting method makes ties virtually impossible for 802.11 exchanges; we have never encountered one.

[4]The interested reader should note that the seemingly plausible "optimization" of moving only along the lowest weight path at each FSM step is incorrect, because subsequent symbols can change which paths are even possible. For instance, with an ACK in the start state, the most likely missing packet is a DATA packet. But if the next packet is a DISASSOCIATION REQUEST with the retry bit on, the missing packet was a DISASSOCIATION REQUEST.

2.2.3 Limitations

While the FSM analysis extracts a great deal of missing information, what can be known with certainty is inherently limited. First, we cannot infer any conversation for which we do not log any packet. Second, for any specific partial conversation, we cannot be certain as to which complete conversation actually transpired. (Selecting the most plausible path leads to our predictions being correct most of the time.) Finally, we cannot always infer all the properties of a missing packet. As a result, the reconstructed trace is most appropriate for deriving measures that are aggregated across many conversations. We show later that the impact of these uncertainties is small, especially given the ability of merging to provide good initial coverage.

We can use several methods to reduce the uncertainty of inferences in the future. For instance, we can better leverage timestamp and sequence number information: missing sequence numbers provide insight into missing conversations, and the transmission times of inferred packets may be estimated if we observe the idle times in the trace and mimic the 802.11 channel acquisition policy. This may lead to more precise inferences, but at the expense of increased complexity (e.g., while most clients increment sequence numbers by one, some have more erratic behaviors). Our current focus is on simpler techniques that bring the most gain.

2.3 Deriving Measures

The enhanced trace generated by merging and inference can be mined in many ways to understand the detailed MAC-level behavior. Many measures, such as packet reception probability, can now be trivially obtained. The trace also facilitates derivations of more sophisticated measures that were not possible before. One such novel analysis that we present below is the estimation of the number of stations that are actively contending for the medium at any given time. This is key to understanding the behavior of the MAC as a function of offered load.

We consider a station to be contending for the medium from the time the MAC-layer gets a packet to send to the time of successful transmission. For packets that require MAC-level responses, transmission is considered successful when the response is successfully received. While the station is contending, there can be multiple retransmissions of the packet and there can be many responses if the initial responses are not correctly received.

The primary difficulty in computing the number of contenders is that judging whether a station is contending requires access to state, such as randomly selected backoff values and carrier sense decisions, that is not present in the trace. We overcome this by making a simple observation: much of the relevant state can be approximated by viewing the station's transmissions through the lens of the medium access rules that it implements. For instance, for 802.11, if we see DATA and DATA$_{retry}$ packets from a station, we know that it was contending for the medium in the time between the two transmissions and at least for some time before the first one.

Our technique is shown in Figure 4. Its description assumes that the reader is familiar with the medium access rules of 802.11. We scan the trace in reverse chronological order and maintain a set of current contenders along with their *idle-wait-time*, which is the amount of idle time they must have waited to acquire the channel before their last observed transmission. Stations are inserted into the set when we observe a packet for them and ejected when that much idle time elapses in the trace. If the transmission is original, i.e., the retry bit is off, the *idle-wait-time* is set to *initial-backoff* to mimic the procedure used by the station itself. If it is a retransmission, we set the *idle-wait-time* to *TIMEOUT* so that the station is eventually ejected from the contenders set even if we do not see another packet from this station (in particular, the corresponding original transmission). Our implementation uses a *TIMEOUT* of 50 ms. The *initial-backoff* is the sum of the 802.11 *DIFS* interval and a randomly selected number of slots between 0 and the initial congestion window.

1: Scan the trace in reverse chronological order and do the following for each packet.
2: Compute the time for which the medium was idle between the current and previous transmissions.
3: Subtract this idle time from the *idle-wait-time* of stations in the contenders set
4: Remove from the contenders set stations with *idle-wait-time* of zero or less.
5: Add the source of the packet (or the destination for response packets such as ACKs) to the contenders set.
6: If the packet is an original transmission, set the station's *idle-wait-time* to *initial-backoff*. Otherwise, set it to *TIMEOUT*.

Figure 4: *Our technique to estimate the number of contenders.*

Our computation of the number of contenders is approximate. First, the exact initial *idle-wait-time* cannot be known, as stations make random choices. Second, by not decrementing *idle-wait-time* when any packet is in the air, we are mimicking that the station senses all transmissions, which of course is not true in practice. Third, we ignore interference effects such as transmissions on other channels. We find that the inaccuracy due to these simplifications is not significant because the initial backoff period they affect is relatively small (Section 4.4); we do not need to approximate the full backoff procedure because that is encoded in stations' subsequent transmission times. Missing packets may cause larger inaccuracies, but their impact is limited by our use of multiple monitors and inference to obtain a reasonably complete trace.

3. IMPLEMENTATION OF Wit

Wit is implemented as three components, halfWit,[5] nitWit and dimWit, corresponding to the three pipeline phases. As a starting point, Wit inserts the traces collected by individual monitors into a database, and then uses the database to pass results between stages. At each stage the trace becomes more complete and gets annotated with additional information. Without the external utilities that they use, halfWit, nitWit and dimWit are roughly 1200, 3200 and 1200 lines of Perl code.

3.1 halfWit: The Merging Component

The vagaries of real data, and its sheer volume, make it challenging to build a robust and fast implementation of the conceptually simple merging process. As one example, we did not expect the 64-bit beacon counters that we use to identify references to roll over. But they do for some APs, perhaps because the APs were reset; we detect such APs and stop using their beacons as references.

For speed, halfWit uses a merge-sort like methodology. It treats the two input traces as time-sorted queues and at each step outputs the queue head with the lower timestamp (after translation). When the precision of timestamp translation is better than half the minimum time to transmit a packet, duplicate packets will appear together as queue heads. Thus, to identify duplicates we only need to compare the queue heads. This is much faster than searching deeper inside the queues for potential duplicates. But it has a subtle interaction with waterfall merging. Consider merging traces from three monitors such that the first and third hear a packet $p1$ and

[5]So-called because it represents about one third of Wit.

the second hears $p2$. If $p2$ is heard at almost the same time as $p1$, the two packets can appear in the merge of the first two traces in any order. Suppose $p2$ comes first. Now suppose that when merging the third trace, its copy of $p1$ is slightly ahead of $p2$ in the merged timeline because of minor imprecision in timestamp translation. The merge-sort based method will fail to merge the two copies of $p1$ because now they do not appear as queue heads together. To address this, we maintain separate logical queues for individual traces in the partially merged trace and then identify duplicates among all the logical heads.

The accuracy of timestamp translation in merging may be affected by monitor processing: monitors must have low variability in the delay between packet reception and timestamp generation. For this reason we use the timestamp applied closest to the hardware from among the several available for the packets. But these timestamps roll over roughly once per hour; we use another, persistent (but more variable) timestamp to detect and correct rollovers.

The precision of the final merge depends on the order in which the monitors are added because order determines the density of common references at each waterfall step. Currently, it has been sufficient to manually order the monitors based on their locations: starting with two close monitors, we successively add monitors close to those already merged. In the future, we will automatically determine the order based on the number of references that monitors have in common with each other.

3.2 nitWit: The Inference Component

nitWit takes the output of halfWit as its input and produces annotated copies of the captured and inferred packets. The critical annotation for each packet is whether it was received. We also piggyback sundry annotations on this processing phase, for instance, we convert the retry-bit field of the 802.11 packet into a counter.

We use a customized regular expression grammar to simplify specification of the FSM. The grammar that corresponds to the simplified FSM in Figure 3 is: [DATA ACK,$,-][DATA$_{retry}$ ACK,$,-]*. The fundamental units, enclosed in square brackets, consist of three fields: a sequence of symbols, an indication of the next step if all the packets represented by the symbols are received, and an indication of the next step if any is dropped. For instance, the first unit above specifies that if both DATA and ACK are received, transition to the accept state (the '$' specifier). This corresponds to the path $Start \rightarrow S1 \rightarrow S2 \rightarrow Accept$ in the FSM. If a packet is not received, transition to the next unit of the regular expression (the '-' specifier). The second unit specifies that what follows is any number of DATA$_{retry}$ and ACK pairs (the '*' specifier). Because it is the final unit, the conversation ends if both packets in any such pair are received. We use *lex*- and *yacc*-like tools to parse this language and generate the FSM. The regular expression for the entire 802.11 protocol is 660 characters long. It produces an FSM with 339 states and 1061 edges. Augmentation adds 15,193 edges.

For speed, we perform two optimizations. Both are guaranteed to not impact the outcome of FSM processing. First, we statically prune some edges in the FSM. If, from a state S_i, a symbol leads to S_j with weight c_{ij} and to S_k with weight c_{ik}, we remove the edge to S_k if there is a path from S_j to S_k with weight c_{jk} such that $c_{ij}+c_{jk} <= c_{ik}$. This eliminates 2,431 augmented edges. Second, we dynamically detect when multiple paths lead to the same state after a transition, and record only the least weight one.

3.3 dimWit: The Derived Measures Component

dimWit operates over the annotated version of captured packets produced by nitWit. Our current implementation does not "merge" captured and inferred packets because the exact timing for the latter is uncertain in some cases.

Along with other measures, dimWit computes the number of contenders in the network. It inserts summary information into a number of auxiliary database tables so that it may compute per contention level measures without reading a number of records proportional to the number of packets. This lets it analyze tens of millions of packets in a few minutes.

4. EVALUATION

We evaluate Wit empirically to understand how well each of its components work. We investigate the following key questions:

(a) What is the quality of time synchronization with merging?

(b) How accurate are inferences of packet reception status?

(c) What fraction of missing packets are inferred?

(d) How accurate is the estimate of the number of contenders?

(e) And finally, to complete the view of network activity, how should we decide between adding monitors and using inference?

Ideally, we would like to answer these questions by comparing our inferences against "ground truth" obtained from the monitored network. But obtaining such authoritative data for deployed networks is intractable. Additionally, instrumentation necessary to obtain the authoritative data is problematic; no commodity hardware to our knowledge reports information on many low-level events required for validation, such as the timing of different retries of a packet. This hinders validation in a testbed as well.

We therefore use simulation as the primary validation method. Because our techniques depend heavily on the MAC layer, we believe that the possibly inaccurate PHY layer models in simulators [12] do not significantly impact our results. But as a sanity check, we test that Wit's results over real traces are self-consistent.

4.1 Simulation Environment

Our simulations involve two APs and forty clients that are randomly distributed on a grid and run 802.11b. We use the QualNet simulator [19] which mimics an 802.11b-like PHY layer; packet reception probability depends on the received signal strength, transmission rate, other packets in flight, and random bit errors. The simulator estimates the maximum radio range in our setup as 480 meters at 1 Mbps and 280 meters at 11 Mbps. To study diverse monitoring environments, we consider three grid sizes: 100x100, 600x600, and 900x900 square meters. The clients generate a mix of web- and DNS-like traffic.

Ten randomly distributed monitors passively sniff the medium and log every packet that they can correctly decode. The timestamp resolution is 1 μsec. Because of the finite precision of timestamps and different propagation delays from the source to different monitors, the timestamps of a packet can differ across monitors. We also generate an authoritative simulation log containing each packet sent and received and when each packet arrived at the MAC layer from higher layers. This log is used to validate the outputs of our tool.

We use the same code base for analyzing simulator and real traces. This lets us check our implementation as well as validate our techniques.

4.2 Merging

To evaluate merging, we check its correctness and characterize the quality of its time synchronization. Both are important to facilitate a wide range of MAC layer studies.

To check if halfWit merges correctly, we use it to merge the views of the monitors in a simulation. For all three grid sizes, we find

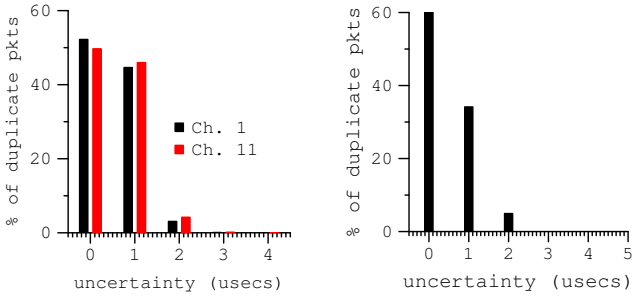


Figure 5: ***The uncertainty of merging, shown as the histogram of uncertainty values.*** **Left:** ***Real traces.*** **Right:** ***Simulator traces.***

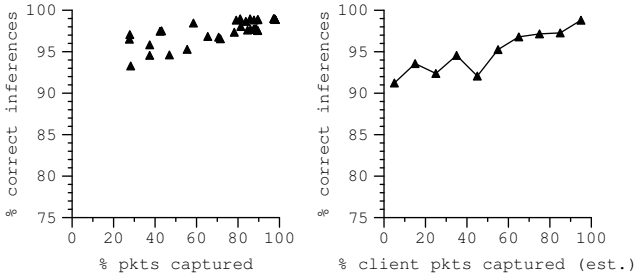


Figure 6: Left: ***The accuracy of inferring packet reception. Each point corresponds to a trace with a certain percentage of captured packets (x-axis); the y-axis shows the percentage of packets whose reception status was correctly inferred.*** **Right:** ***The accuracy of inferences as a function of*** nitWit***'s estimate of the percentage of client's packets captured. The y-axes start at 75%.***

using the authoritative log that all duplicates and only duplicates are removed. Because clock behaviors in the real and simulated environments likely differ, this is not a litmus test. But, along with manual verification of merged real traces, it boosts our confidence in the tool's implementation and its output with real traces.

Next, we evaluate synchronization quality, which tests the robustness of our timestamp mapping method to variabilities in real clocks. We use real traces from a live network (described in detail in the next section) for this experiment. It requires that quality be measured without knowledge of ground truth. We accomplish this by using the difference in the translated timestamps of the packets that are identified as duplicates during the merge. This is a measure of timestamp uncertainty. The minimum uncertainty value is zero, for perfect synchronization. The maximum is 106 μsecs (half of the minimum time to transmit an 802.11b packet), since duplicate identification is limited to packets within that threshold. Each pair of identified duplicates at each waterfall step produces one uncertainty value; we study the distribution of the values. Obtaining 106 μsecs for even a small fraction of values suggests an incorrect merge, as there are probably unidentified duplicates beyond the threshold.

Figure 5 plots the histogram of uncertainty values. The left graph is for real traces from Channels 1 and 11 which have four and five monitors, respectively. For both, the merge is very precise. The 99.9 percentile uncertainty is 2 μsecs. The worst is 8 μsecs (not in the graph). For comparison, the uncertainty of merging the simulator traces is shown on the right. Due to possibly different timestamps on identical packets across monitors, rather than it being zero, the 99.9 percentile uncertainty of even simulator merges is 2 μsecs. This suggests that potential inaccuracies of real clocks do not significantly increase the uncertainty of merging.

At 2 μsecs, the uncertainty in merged timestamps is smaller than the slot time of 802.11b (20 μsecs). This enables a class of inferences that are otherwise not possible. For instance, consider two packets are in flight simultaneously: we can distinguish a collision in which the two sources start in the same slot from a failure of carrier sense in which one source does not sense the other.

We now study the relationship between the quality of time synchronization and the frequency of common references. To do so, we compute uncertainty of merging two real traces for reference periods of 1, 10, 100, 1000, and 10,000 seconds. A period of 10 seconds means that the successive references used for time translation are spread roughly 10 seconds apart; we ignore intermediate references. The 99.9 percentile uncertainty is 2 μsecs for 100 seconds or less, 18 μsecs for 1000 seconds, and 106 μsecs for 10,000 seconds. Given that APs send beacons roughly every 100 ms, this implies that the uncertainty can be kept down to 2 μsecs as long as the two traces have in common at least 0.1% of the beacons from at least one AP.

To compare our technique with that of Yeo *et al.* [27], we use their method to merge two real traces of different lengths. We find that the uncertainty increases with length. The 99.9 percentile value is 5 μsecs for 1-hour traces, 12 μsecs for 2-hour traces, and 106 μsecs for 4-hour traces. (These results are better than those reported by Yeo *et al.*, who obtain a 40-μsec uncertainty for two 12.5-minute traces.) The last merge is likely incorrect.

4.3 Inference

To evaluate nitWit, we run it over simulator traces and study its ability to infer packet reception statuses and missing packets.

The left side of Figure 6 shows how accurately nitWit infers whether packets were received. Correct inferences are shown as a function of the percentage of the total packets captured in a trace. We obtain traces with different capture percentages by using different monitors and merge combinations. Correctness and capture percentages are computed using the authoritative simulator log.

We see that nitWit is quite accurate: its accuracy is 95% even when the trace contains only half of the total packets and 90% even when it contains only a third. In our data, a common scenario in which nitWit is relatively less accurate is when it observes one or more ACKs without corresponding DATA packets; the ACKs by themselves yield little information about their reception.

Interestingly, we find that nitWit can estimate when its inferences will be relatively less accurate. This is because the accuracy of the inferences for a client depends on the fraction of the client's packets that were captured. This fraction can in turn be estimated from traces without knowledge of ground truth as a side-effect of how many missing packets are inferred. The capture estimate we compute is the ratio of the number of packets captured for the client to the sum of the packets captured and inferred for the client.

The right side of Figure 6 shows how accurately nitWit infers receptions for a client's packets as a function of this capture estimate. Clients are binned by their capture estimate into 10%-wide bins, and the average accuracy of the bin is plotted as the y-value. Overall, nitWit does well even for clients from whom a small fraction of packets are captured. This is because the monitors often capture the other end of the conversation. We see that accuracy decreases with the estimate of packets captured. This enables a user of nitWit to judge the accuracy of inferences for a client and, if need be, focus on clients with accurate inferences.

Next, we study the ability of nitWit to complete a trace by inferring missing packets. Figure 7 plots the percentage of packets that are either inferred or captured versus the percentage captured.

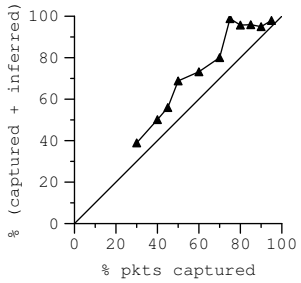


Figure 7: ***The ability of*** Wit ***to infer missing packets.***

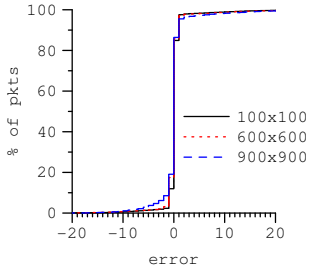


Figure 8: ***The CDF of error in estimating the number of contenders, for all three grid sizes.***

Traces are binned into 5%-wide bins based on their capture percentage; the y-values are the averages for the bin. We see that nitWit adds roughly 10-20% packets when the capture percentages are low and that directly capturing 80-90% of the packets lets us infer much of the remaining ones. In this role, nitWit is especially useful when the monitors hear only one end of the conversation well, as it can then infer the other end. This simplifies passive monitoring: we can engineer placements that use fewer monitors to capture one end of each conversation well, instead of both ends.

We also ran a self-consistency check over real traces to build confidence that PHY layer losses in real environments do not lead to significantly inaccurate inferences. Specifically, we infer packet reception over two traces; the second trace is obtained by merging the first with another trace and has 21% more packets. If realistic PHY layer loss patterns affect our inferences significantly, the inferences over the two traces are likely to differ significantly. We find that our packet reception inferences are consistent for 93% of the packets that are present in both traces. This is encouraging; if this consistency reflects correctness, then nitWit takes us from having no reception status information to correct inferences for the vast majority of the cases, even when many packets are missing from the input trace.

4.4 Estimating Contenders

To evaluate our estimate of the number of contenders, we run dimWit on the merged traces of all ten simulated monitors and compare the estimate against the authoritative log. Figure 8 plots the CDF of error in our estimate at the end of each transmission. Error is computed as the estimated minus the actual number of contenders. We see that our estimates are quite accurate overall. The accuracy decreases slightly with grid size because fewer packets are captured. In the worst case of the 900x900 grid, where 90% of the packets are captured, dimWit is within ± 1 for 87% of the cases and ± 2 for 92% of them. In the 100x100 grid, where 98% of

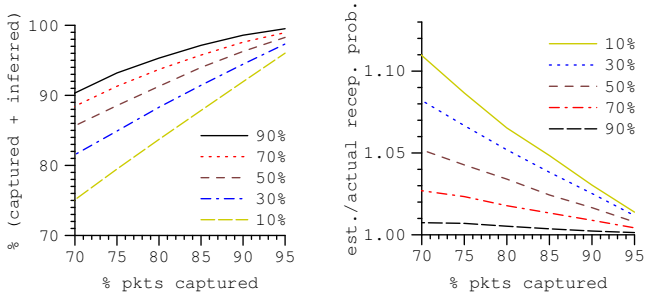


Figure 9: ***Effectiveness trends of inference. The x-axis is monitor capture probability. The curves represent different round-trip reception probabilities.*** **Left:** ***The ability to infer missing packets.*** **Right:** ***The accuracy of the estimate of reception probability.***

the packets are captured, our estimates are within ± 1 for 95% of the cases. Closer inspection of data reveals that high error values tend to correspond to cases with a high number of contenders (e.g., estimating 15 or 25 contenders when there are 20) because the actual number changes rapidly. This reduces the relative error in our estimates.

We find that our estimates are largely insensitive to the exact choice of the initial-backoff parameter within a reasonable range. We varied the parameter value between zero and twice the initial congestion window and observe that our accuracy is not significantly impacted. For instance, with a value of zero, our estimates are within ± 1 for 82% of the cases in the 900x900 grid. This is encouraging because most of the approximation in our computation occurs in the initial backoff phase (because stations' transmissions do not reflect their choices in this phase). For instance, we approximate that stations defer to all transmissions in the initial phase. Interestingly, the value of zero approximates the other extreme in which stations do not defer to any transmission in this phase, and even then our estimates are reasonably accurate.

4.5 Inference Versus Additional Monitors

Given that both inference and additional monitors help to complete our view of network activity, how should these two methods be combined in a practical system? To understand this, we study a simple model that exposes the ability of inference to deal with incomplete data. We generate artificial traces in which the monitor capture probability and the node reception probability are held constant. Clients repeatedly attempt to send data. Each DATA packet, both original and retried, and ACK is independently logged with the capture probability. Additionally, we drop packets according to the specified round-trip reception probability, divided equally in each direction. We vary the capture probability from 0.7 to 0.95; higher values represent setups with more monitors. We vary round-trip reception probability from 0.1 to 0.9 to cover a range of conditions. This experiment lets us isolate capture and reception probabilities in an understandable setting.

Figure 9 shows the results of running nitWit over the traces. In both graphs, the x-axis is capture probability, and curves represent different round-trip reception probabilities. The left graph plots the percentage of packets that are either captured or inferred. The right graph plots the ratio of nitWit's estimate of reception probability to the actual value. This shows the accuracy of inferences over the trace. The main conclusion we draw is that merging and inference are complementary. At low capture probabilities, while nitWit sub-

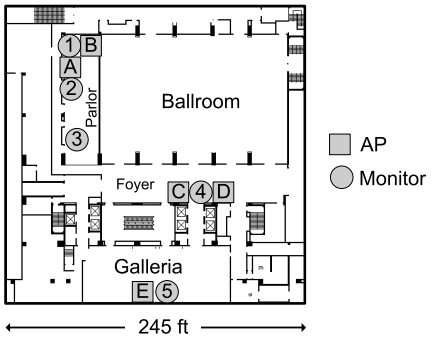


Figure 10: ***Monitoring environment at the conference. The layout shows the approximate locations of the APs and the monitors. Ballroom hosted the main conference and had only limited wireless access on the back and left. Parlor acted as the terminal room and was most active. Galleria hosted the workshops.***

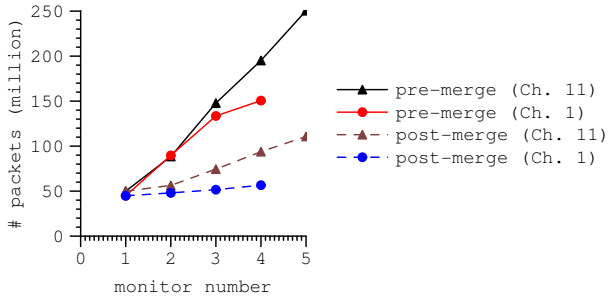


Figure 11: ***Cumulative packet counts as monitors are merged. "Pre-merge" counts the total number of packets, without removing duplicates. "Post-merge" represents the merged traces.***

stantially adds to the trace, the right course of action is to add monitors to improve the underlying capture probability. Assuming that the monitors log each packet with an independent probability, additional monitors in this regime will be highly effective at adding to the trace. There are diminishing returns as monitors become more dense and the capture probability is already high. Here, nitWit almost completes the trace with missing packets that would be hard to recover through additional monitors, especially for well-connected clients. Similarly, while the accuracy of inferences is high over the entire range of capture probability, it is especially good in the range that represents good coverage by the monitors. Above 85% capture probability, the relative error in the estimates is less than 5%, and the absolute error is even lower.

5. APPLYING Wit TO A LIVE NETWORK

We now report on our experience in applying Wit to analyze a live wireless network. By necessity, we focus on a few analyses; it is straightforward to perform many others.

5.1 Monitoring Environment

Our wireless environment is the SIGCOMM 2004 conference which spanned four days and had roughly 550 attendees. We view this as a large, busy setting. The layout of the conference floor of the hosting hotel is depicted in Figure 10. The official wireless network of the conference had five APs. Except for AP D, which operated on Channels 6 and 8, the APs operated on Channels 1 and 11. Some of the APs switched channels during the conference. Internet connectivity was enabled through four separate DSL access lines. In addition to the conference network, a number of transient infrastructure and ad hoc networks were present, and the hotel had its own, private wireless network on Channel 6. In our view, the diversity and transience of clients makes it intractable to study this environment using instrumentation.

We passively monitored this network using five PCs, each with three wireless NICs whose external antennae were placed at least a foot apart. Two NICs of each monitor listened on Channels 1 and 11, and the third listened on Channel 6 or 8. Monitors logged all observed activity, including control, management and data packets. Complete MAC headers and PHY information, such as transmission rate, were logged for each packet. All monitors except 4, which was switched off and stored elsewhere at nights, were active for the entire duration of the conference. We analyze traces from Channels 1 and 11.

The SIGCOMM 2004 wireless network had intermittent usability problems. We understand that these stemmed from DHCP and DNS issues and disrupted Internet connectivity for some clients. We believe that these problems do not affect the underlying MAC behavior which is our focus. They do, however, lower the average load on the network during connectivity disruptions.

5.2 Merging with halfWit

We merged the traces from Monitors 1-4 on Channel 1 and from all five monitors on Channel 11 to produce a merged trace for each channel. Monitors were merged in the order of their number. The Monitor 5 trace of Channel 1 did not have enough references in common with the merged trace of the other four monitors for it to be correctly merged. We exclude it from our analysis.

Our experience provides a useful lesson on the placement of monitors. A natural tendency is to place monitors far apart to maximize the capture of unique packets. But this minimizes the overlap between monitors. Placement that yields too little overlap is a poor strategy because it hinders merging.

To understand the value of merging in enhancing the view of wireless activity, we count the number of additional packets and clients that are present in the merged trace compared to the Monitor 1 trace. We find that merging adds 28% packets and 12% clients on Channel 1 and adds 124% packets and 37% clients on Channel 11. In addition to more overall activity, the additional packets represent enhanced views of individual clients: the merged trace has 12% and 60% more packets per client on the two channels.

Figure 11 shows the gains of merging in more detail. It plots the cumulative number of packets as additional monitors are merged. The solid curves show the number of packets before duplicates are removed, and the dashed ones show the merged traces (after removing duplicates). There is significant overlap in what the monitors hear, yet each additional monitor increases the number of unique packets in the trace. This is true even when we merge monitors 1 and 2 that sit next to each other. Thus, even a dense array of monitors may miss packets. This motivates the need to infer missing packets, as we do with nitWit, because capturing this information through monitoring alone is almost impossible.

5.3 Inference with nitWit

We applied nitWit to the two merged traces. The Channel 1 merge has 56M packets of which 49M are unicast. nitWit processed 30M conversations with 26K distinct packet sequences. The top three sequences were DATA–ACK (51%), BEACON (22%), and DATA–

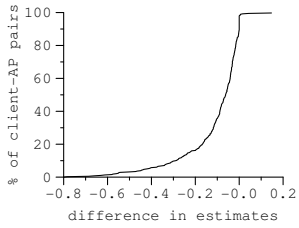


Figure 12: ***The CDF of the difference in the reception probability estimates of*** nitWit ***and the heuristic based on the retry-bit.***

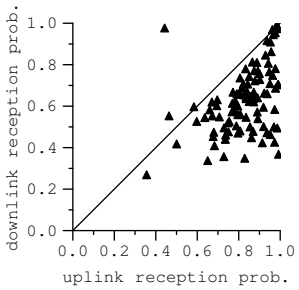


Figure 13: ***The uplink versus downlink reception probability for clients in the network, computed over the entire Channel 1 trace.***

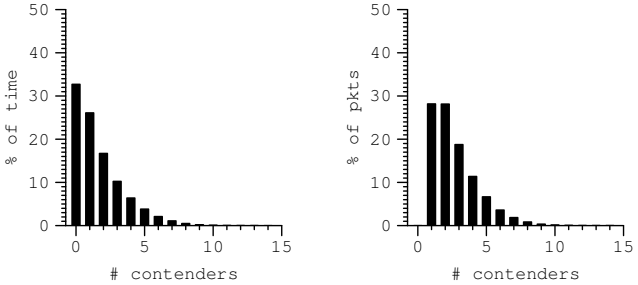


Figure 14: ***The prevalence of different levels of contention, computed over a busy hour-long interval of the Channel 1 trace. The graphs show the percentages of total time spent (left) and packets sent (right) at each contention level. The*** y***-axes end at 50%.***

DATA$_{retry}$–ACK (6%). nitWit inferred that 80% of the unicast packets were received by their destination and inferred an extra 5.5M packets; we estimate from this that the monitors captured 90% of the total packets. The Channel 11 merge has 111M packets of which 95M are unicast. nitWit inferred that 94% of the unicast packets were received, and inferred an extra 24M packets with a corresponding capture estimate of only 79%. Therefore, while the view of Channel 1 is reasonably complete, the view of Channel 11 is poorer, and we expect the measures we compute for it to be less accurate.

The traces also let us assess how well nitWit can infer various properties of the missing packets. For the Channel 1 merge, we count the inferred packets for which the exact size, transmission time and transmission rate could be reconstructed. We find that size, time and rate can be inferred for 76%, 64% and 42% of the packets, respectively, and that all three properties can be inferred for 34% of the packets. The low percentage of rate inferences is because the rate of any other packet in the conversation provides little information about the rate of a missing data packet (which could be improved by inferring the rate adaptation behavior of clients.)

We observe that nitWit can lead to simpler and likely more accurate estimates. This is because it systematically extracts latent information from the traces, whereas the heuristics that must otherwise be used are based on simpler, less complete, models [11, 21]. Consider two cases:

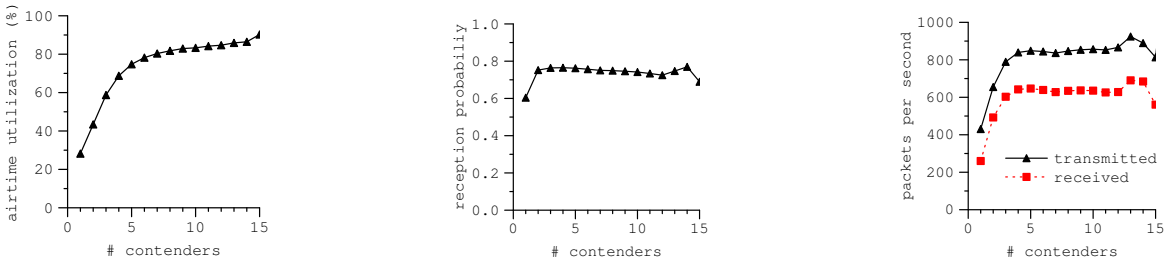
1. In earlier work, we estimated the reception probability of packets between clients and APs based on the retry bit [21]. Each data packet with the retry bit set suggests that the earlier transmission was lost. As a heuristic, we estimated reception probability as one minus the fraction of data packets with the retry bit set. Figure 12 plots the CDF of the difference between this heuristic and the reception probability computed using nitWit's reception inference. There is a data point for both directions of each client-AP pair that exchange over 100 packets. We see that the heuristic computes significantly lower reception probabilities, by 0.20 for 15% of the cases and by 0.10 for 30% of them. While we cannot be certain without ground truth, we believe that the Wit estimates are closer to the correct values based on validation checks with the simulator. The heuristic is biased by assumptions that do not hold: it assumes that both monitor capture and reception probability are independent of factors such as size, rate and type; in practice, data packets are more likely to be lost than ACKs due to their bigger size and original data packets are more likely to be lost than retries due to their higher average transmission rate. There appears to be no straightforward way to remove these biases from the heuristic.

2. The monitor capture percentage can be estimated with a heuristic based on the DATA-ACK exchange. Each ACK without a corresponding data packet indicates that a data packet was not captured. An estimate of the capture ratio is then the number of data packets in the trace divided by the sum of this number and the number of ACKs without a corresponding data packet [11, 21]. We obtain capture percentages of 94% and 85% with this heuristic for the merged traces of Channel 1 and 11, respectively. The corresponding nitWit estimates stated above are 90% and 79%, respectively. Again, while we cannot be certain without ground truth, the heuristic seems to overestimate capture. This is presumably because it does not account for other patterns of missing packets, e.g., missing ACKs or DATA-ACK pairs, whereas our FSM analysis can account for such patterns using subsequent packets in the conversation.

5.4 Analysis with dimWit

We now present sample analyses of Channel 1 of the SIGCOMM 2004 wireless environment with dimWit. We present a series of 802.11 operational insights that are enabled by Wit, not simply by passive monitoring. These observations could not have been gathered via simulation or testbeds either because they depend on 802.11 usage in real environments.

Uplink was more reliable than downlink Figure 13 compares the reception probability for uplink (to the AP) versus downlink transmissions for 100 randomly selected clients. The reception probabilities are often asymmetric and the uplink is usually more reliable. We are surprised by this result; we expected the downlink direction to be more reliable because APs tend to transmit at a higher power, boosting the chances of correct reception. Additionally, fewer packets from the AP should be involved in collisions because all clients should be able to hear the AP even though they may not all be able to hear each other. We speculate that the uplink was more reliable because commercial APs have better, possibly multiple, antennae that improve their decoding ability.



(a) airtime utilization (b) reception probability (c) throughput (pkts)

Figure 15: ***Various measures as a function of the number of contenders in the network, computed over the entire Channel 1 trace.***

Offered load was mostly low Figure 14 shows histograms of each contention level for an hour long busy interval which had 260 unique clients. The left graph plots the percentage of *time* spent at each level. We see that most of the time there are no nodes waiting to transmit. The time spent at higher contention levels decreases exponentially such that there are more than five contenders less than 4% of the time. The right graph plots the percentage of *packets* sent at each contention level. We again see that the system is exercised most at low contention levels. (Of course, no packets are sent at zero contenders.)

While the prevalence of low contention is perhaps not surprising, it is difficult to reason about offered load without the analyses provided by dimWit. For instance, utilization per unit time plots (e.g, those in earlier work [11, 21]) do not distinguish between low utilization due to a no-contender scenario and low utilization due to a high-contender scenario in which there are backed-off stations. Similarly, active clients per unit time plots do not convey information as to whether the clients contend simultaneously.

The medium was inefficiently utilized Figure 15(a) plots airtime utilization as a function of the number of contenders. Utilization is computed as the percentage of time for which the medium was busy with at least one transmission. The ending time of a packet is its timestamp, and the starting time is computed using its size and transmission rate and the length of the 802.11 (long) preamble. Inter-frame spaces are not included in this computation. We see that the medium is poorly utilized in the common case of few contenders. This is surprising because it is not the case that the medium is idle due to a lack of offered load: by definition there is at least one station waiting to transmit at each non-zero contention level. As reference, the theoretical utilization of a single node sending 500 byte data packets at 5.5 Mbps and receiving ACKs at 2 Mbps (with no losses) is roughly 75% rather than 30% as we observe. Thus, it appears that nodes often wait unnecessarily in backoff phases before they transmit.

Reception probability did not decrease with contention Figure 15(b) plots the packet reception probability as a function of the contention level. We expected a decline with the number of contenders due to increased collision losses. Instead, we were surprised to find that reception probability remains steady.[6] This suggests that inherent "radio losses" were the dominant cause of packet drops. Consistently, we find that only 0.45% of the packets in our trace had their transmissions overlap with another packet.

Performance was stable at high contention levels Figure 15(c) shows the rates of packets transmitted and received (by the destination) in the network. This is the throughput of the network. It initially increases with the number of contenders and then stabilizes at five or more contenders. As reference, the throughput with a single node sending 500 byte data packets at 5.5 Mbps and receiving ACKs at 2 Mbps (with no losses) will be roughly 1200 packets per second. Thus, we find the MAC operates well at high contention levels. Contrary to a concern of recent work [11, 21], we do not observe throughput decreases due to undesirable interactions with transmission rate adaptation (where high contention leads to lower transmission rates because of losses).

Taken together, our observations suggest that the 802.11 MAC is tuned for higher contention levels than those we observe in our measured network. It assumes that most losses are due to contention and hence backs off in response to any loss; in reality, most losses appear to be radio losses that do not warrant backoff. The MAC appears overly biased towards avoiding collisions by using larger than necessary backoff intervals. The result is that the medium usage is quite inefficient for the common case of low offered load. This suggests that there is an opportunity for a MAC that adapts to its environment to be efficient at low load as well as high load. Indeed, recent work explores some aspects of this problem [9].

6. RELATED WORK

Most work on measurement-driven analysis of wireless networks either uses traces from the wired portion of the network and SNMP logs of APs [4, 5, 7, 8, 23, 26] or uses instrumentation in testbed settings [2, 10, 29]. While these approaches provide useful insights into the behavior of wireless networks, the information they gather cannot be used to study the detailed behavior of the MAC layer in deployed networks.

A few recent works have explored passive monitoring of wireless networks. Jardosh *et al.* [11] and Rodrig *et al.* [21] analyze data collected by individual monitors at live networks to analyze aspects such as 802.11 MAC overhead, airtime utilization, and congestion. But the view of a single monitor is inherently limited. To overcome this limitation, Yeo *et al.* originally proposed merging the views of multiple monitors [27]. Our merging approach is built on their observation regarding reference packets, though our exact technique is different and more precise. We view our work as posing passive monitoring as a broad approach to analyze live wireless networks and advancing the state of the art through novel techniques to infer, for instance, the reception status of packets and the number of stations competing for the medium.

Jigsaw is a concurrently developed system to perform cross-layer analysis of enterprise wireless networks [6]. While it focuses on a different problem, like Wit, it builds on merging and inferring the reception status of packets. However, its techniques are different.

[6]The increase in reception probability from one to two contenders might be because poorly connected clients dominate the former case.

Jigsaw focuses on merging a large number of traces in real-time, and, in contrast to our formal approach, its inferences are based on an ad hoc set of rules that encode common protocol exchanges.

Our view of traces as sentences from a formal language is inspired by work on using traces to test protocol implementations [14]. Our goal is different, however, and to our knowledge, we are the first to use formal methods to add missing packets to the trace and infer packet reception statuses.

7. CONCLUSIONS

We presented Wit, a non-intrusive tool that builds on passive monitoring to support detailed MAC-level analysis of operational 802.11 wireless networks. It uses several novel techniques to enhance the necessarily incomplete view of system activity obtained through passive monitoring. First, it merges the independent views of multiple monitors into a single, consistent view. Then, it uses an engine based on formal language techniques to infer packets that were missed by all monitors as well as infer which packets were received by their destinations. Finally, it derives detailed performance measures. We provided a procedure to estimate offered load in terms of the number of nodes competing for the medium at any given time. We used a mix of simulation and real traces to evaluate our techniques, with encouraging results.

To demonstrate its abilities, we applied Wit to a multi-monitor trace of a live network, performing analyses that would otherwise not be possible or rely on less accurate heuristics. We uncovered a picture of MAC operation that warrants further study. For instance, we found that our network predominantly had low contention and that the medium was inefficiently utilized during those times. It appeared that the MAC was tuned for the uncommon case of high contention levels, e.g., it backs off more than necessary in response to any loss.

Our work is a nascent step towards non-intrusive methods that can provide deep analyses of the behavior of operational 802.11 wireless networks. Our formal language based inference engine is both general (e.g., we have used it to discover protocol violations) and highly effective in extracting latent information from traces. It will only improve as we use it to model and infer other behaviors. It may be applicable in settings beyond 802.11 as well. Our procedure for estimating the number of contenders highlights new kinds of analyses that can extract higher-level information from packet-level wireless traces. We hope that future research in this area will lead to a better understanding of 802.11 "in the wild."

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for feedback on the submitted draft of this paper. This work was supported in part by the NSF (Grants CNS-0133495 and CNS-0338837).

9. REFERENCES

[1] IEEE Std. 802.11i – Amendment 6: Medium access control security enhancements, 2004.

[2] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *SIGCOMM*, 2004.

[3] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh. A comparison of overlay routing and multihoming route control. In *SIGCOMM*, 2004.

[4] A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan. Characterizing user behavior and network performance in a public wireless LAN. In *SIGMETRICS*, 2002.

[5] M. Balazinska and P. Castro. Characterizing mobility and network usage in a corporate wireless local-area network. In *MobiSys*, 2003.

[6] 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 *SIGCOMM*, 2006.

[7] F. Chinchilla, M. Lindsey, and M. Papadopouli. Analysis of wireless information locality and association patterns in a campus. In *INFOCOM*, 2004.

[8] T. Henderson, D. Kotz, and I. Abyzov. The changing usage of a mature campus-wide wireless network. In *MobiCom*, 2004.

[9] M. Heusse, F. Rousseau, R. Guillier, and A. Duda. Idle sense: An optimal access method for high throughput and fairness in rate diverse wireless LANs. In *SIGCOMM*, 2005.

[10] K. Jamieson, B. Hull, A. Miu, and H. Balakrishnan. Understanding the real-world performance of carrier sense. In *workshop on Experimental Approaches to Wireless Network Design and Analysis (E-WIND)*, 2005.

[11] A. Jardosh, K. Ramachandran, K. Almeroth, and E. Belding-Royer. Understanding congestion in IEEE 802.11b wireless networks. In *IMC*, 2005.

[12] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions. In *symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, 2004.

[13] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. An experimental study of delayed Internet routing convergence. In *SIGCOMM*, 2000.

[14] D. Lee, A. N. Netravali, K. K. Sabnani, B. Sugla, and A. John. Passive testing and applications to network management. In *ICNP*, 1997.

[15] S. Mangold, Z. Zhong, G. R. Hiertz, and B. Walke. IEEE 802.11e/802.11k wireless LAN: Spectrum awareness for distributed resource sharing. *Wireless Communications and Mobile Computing*, 4(8), 2004.

[16] A. Miu, H. Balakrishnan, and C. E. Koksal. Improving loss resilience with multi-radio diversity in wireless networks. In *MobiCom*, 2005.

[17] V. Paxson. Automated packet trace analysis of TCP implementations. In *SIGCOMM*, 1997.

[18] V. Paxson. End-to-end routing behavior in the Internet. In *SIGCOMM*, 1997.

[19] Qualnet network simulator by Scalable Network Technologies. http://www.qualnet.com.

[20] I. Ramani and S. Savage. Syncscan: Practical fast handoff for 802.11 infrastructure netwroks. In *INFOCOM*, 2005.

[21] M. Rodrig, C. Reis, R. Mahajan, D. Wetherall, and J. Zahorjan. Measurement-based characterization of 802.11 in a hotspot setting. In *workshop on Experimental Approaches to Wireless Network Design and Analysis (E-WIND)*, 2005.

[22] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The end-to-end effects of Internet path selection. In *SIGCOMM*, 1999.

[23] D. Schwab and R. Bunt. Characterising the use of a campus wireless network. In *INFOCOM*, 2004.

[24] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *SIGCOMM*, 2002.

[25] G. Tan and J. Guttag. The 802.11 MAC protocol leads to inefficient equilibria. In *INFOCOM*, 2005.

[26] D. Tang and M. Baker. Analysis of a local-area wireless network. In *MobiCom*, 2000.

[27] J. Yeo, M. Youssef, and A. Agrawala. A framework for wireless LAN monitoring and its applications. In *workshop on Wireless Security (WiSe)*, 2004.

[28] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of Internet flow rates. In *SIGCOMM*, 2002.

[29] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SenSys*, 2003.