To appear in Proceedings of ACM SIGCOMM'97, Cannes, France, September, 1997

Trace-Based Mobile Network Emulation

Brian D. Noble,[*] M. Satyanarayanan,[*] Giao T. Nguyen,[†] Randy H. Katz[†]

[*]Carnegie Mellon University
School of Computer Science
Pittsburgh, PA
{bnoble,satya}@cs.cmu.edu

[†]University of California, Berkeley
Department of Electrical Engineering and Computer Science
Berkeley, CA
{gnguyen,randy}@cs.berkeley.edu

Abstract

Subjecting a mobile computing system to wireless network conditions that are realistic yet reproducible is a challenging problem. In this paper, we describe a technique called *trace modulation* that re-creates the observed end-to-end characteristics of a real wireless network in a controlled and repeatable manner. Trace modulation is transparent to applications and accounts for all network traffic sent or received by the system under test. We present results that show that it is indeed capable of reproducing wireless network performance faithfully.

1 Introduction

How does one subject a mobile computing system to realistic yet reproducible wireless networking conditions? Reproducible behavior is important for three reasons. First, it is essential for thorough evaluation of the performance of a mobile computing system. Second, it is necessary for comparative evaluations of alternative system designs. Third, it is valuable in debugging mobile systems because it enables the re-creation of conditions that trigger rare but serious bugs. Unfortunately, obtaining repeatable performance is extremely difficult because the quality of wireless networks can vary dramatically and unpredictably over time and space [6, 12].

In this paper we describe a solution, called *trace modulation*, which combines three distinct ideas. First, performance of a real, wireless network is captured through trace collection. Second, these complex network observations are reduced to a list of parameters of a simple, time-varying network model. Third, the network performance described by these parameters is reproduced in a controlled manner. Applications run unmodified in the resulting emulated network, and experience end-to-end performance mirroring the original, physical network.

This research was supported by the Air Force Materiel Command (AFMC) under DARPA contract number F196828-93-C-0193, the Army Communications Command (CECOM) under DARPA contract number DAAB07-95-C-D154, and the State of California MICRO Program. Additional support was provided by AT&T, Hughes Aircraft, IBM Corp., Intel Corp., and Metricom. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFMC, CECOM, DARPA, AT&T, Hughes, IBM, Intel, Metricom, Carnegie Mellon University, the University of California, the State of California, or the U.S. Government.

Trace modulation differs from superficially similar approaches such as trace replay and trace-driven simulation in that it creates a *synthetic networking environment* rather than a *synthetic workload.* This synthetic environment is transparent to applications and the network protocol stack above the data link layer. Further, it accounts for all network traffic sent or received by the modulated host, not just that of specific applications.

We show that trace modulation accurately reproduces the original wireless environment. Our validation experiments consist of three benchmarks — Web browsing, FTP transfers, and the Andrew Benchmark [7] — run in four different WaveLAN [2] wireless networking scenarios. In the few cases where the measurements indicate significant divergence, our analysis suggests that the use of fine-granularity, low-drift, synchronized clocks would substantially improve accuracy. Unfortunately, such clocks are not yet readily available on mobile platforms.

Section 2 explains why mobile system evaluation is difficult, introduces our methodology, and compares it to related work. Section 3 describes the details of our implementation. Our strategy for validation is laid out in Section 4, and its results appear in Section 5. We conclude in Section 6 with a summary of our work, and a discussion of its potential for broader application.

2 Background and Overview

2.1 Experimental Control in Mobile Systems

There are numerous difficulties in evaluating mobile, wireless systems. Unlike wired networks, the medium over which wireless messages travel is difficult to isolate. Mobile clients which are not part of the evaluation may perturb results by polluting the wireless spectrum. The shared use of unlicensed spectrum by wireless networks and devices such as cordless phones is a further challenge to experimental control.

Even if the relevant region's wireless spectrum could be isolated, experimental control remains a challenge. Wireless propagation is affected by environmental factors that are both spatially and temporally dependent. Further, precise duplication of physical motion is very difficult. Good experimental control is nearly impossible to achieve under these conditions, and interpretation of results is complicated by the presence of many confounding factors.

Our solution to this problem is to provide an emulation environment that is controlled and repeatable while also being faithful to the real mobile environment under consideration. To do this, we use a *trace-based* approach, recording the performance of a real wireless deployment and reproducing that performance in a controlled manner. Unlike most trace-based systems, modulation influences the environment in which a system operates rather than generating the workload for that system. In other words, we create a synthetic environment in which to execute a real workload, rather than create a synthetic workload in a real environment.

While trace modulation is especially valuable for wireless networks, it can also be useful in other contexts. In general, it is difficult to reproduce the precise behavior of live networks supporting complex distributed systems. Our methodology offers the potential for improved reproducibility in evaluating such systems.

Of course, trace modulation is not a panacea. A single trace can only capture a snapshot of the varying performance along a particular path. Further, even trace-based evaluation cannot offer precisely reproducible results because network processes are nondeterministic. In spite of these limitations, our approach strikes an excellent balance: it comes close to the realism of live experiments, while preserving much of the flexibility, ease of use, and reproducibility of simulation.

2.2 Methodology

Our methodology consists of three phases: *collection, distillation,* and *modulation.* In the collection phase, an experimenter with an instrumented mobile host physically traverses some path. During the traversal, packets from a known workload are generated. The mobile host records observations of these packets as well as network device characteristics. At the end of the traversal, the list of observations represents an accurate trace of real network behavior. By performing multiple traversals of the same path, one can obtain a trace family that captures network quality variation on that path.

The distillation phase transforms a collected trace into a form suitable for modulation. For each instant in time, distillation examines the performance of the known workload and produces a set of parameters for a simple network performance model. By composing these, distillation produces a concise, time-varying description of network performance.

In the modulation phase, mobile system and application software is subjected to the network behavior described by the distilled trace. Unmodified software is run on a LAN-attached host whose kernel has been extended to read the trace and to delay or drop packets in accordance with its model parameters. The mobile software thus experiences a network environment indistinguishable from that recorded in the trace, but experiments can be carried out without mobile network hardware or physical motion. It is important to note that trace modulation is fully transparent to mobile software — no source or binary changes have to be made, and all network traffic into and out of the mobile host is accounted for.

2.3 Related Work

The original contribution of our methodology lies in its synthesis of three distinct ideas:

- trace collection to accurately capture observed network behavior;
- reduction of observations into a time series of parameters for a simple network model;
- application-transparent network emulation through model-driven delays and losses of packets in a layer below the API of an operating system.

We are not aware of any previous work, published or unpublished, that combines these ideas in a similar manner. However, each aspect of our methodology has been investigated in isolation by other researchers.

The best-known system for network tracing is the Berkeley Packet Filter [11], which is typically used in conjunction with `tcpdump` [9]. This architecture is efficient and flexible, and has rightly found great favor with the networking community. Our trace collection mechanism differs from the Berkeley Packet Filter in that we record device characteristics in addition to information from packets. While not strictly necessary for trace modulation, such a record of device behavior in actual use is valuable for a better understanding of wireless networks [12].

The notion of reducing complex network observations to simple parameters through controlled workloads is commonly used in modelling physical channels. Our determination of bottleneck bandwidth is quite similar to the packet-pair approach used by Keshav [10], but our workload enables derivation of additional network parameters.

The network emulation package most similar to our modulation kernel is *hitbox*, which was used in evaluating the performance of TCP Vegas [1]. Unlike trace modulation, hitbox models networks with relatively static performance. A more flexible system, the Probe/Fault Injection Tool [4], allows any protocol layer to be encapsulated by a lower layer to perturb existing traffic, and a higher layer to generate test traffic. However, these layers are driven only by synthetic models, not by empirically derived ones. The Lancaster emulator [3] uses a central server rather than an emulation layer in each host.

More broadly, the use of user-level libraries for network emulation is widespread. Examples include Delayline [8] and the `slow` mechanism of RPC2 [16]. While useful, such libraries have two shortcomings: they require recompilation or relinking of applications, and they only influence traffic to or from the applications in question.

3 Design and Implementation

3.1 Collection Phase

There are two key issues in trace collection: what to collect and how to collect it. We have defined a trace format [13] that is flexible and extensible while remaining fully self-descriptive. We collect both packet traffic information and characteristics of mobile network devices; our collection platform incurs modest overhead, and reliably detects lost data.

3.1.1 Data Collected

Trace collection logs every outgoing and incoming packet, along with the time at which it was sent or received. Where relevant, we also collect protocol-specific information such as sequence numbers, flags, and destination and source ports.

Our known workload, a modified version of the `ping` utility, consists of ICMP ECHO and ECHOREPLY packets. For ECHO packets, we collect the *id* field, which is the process id that generated the ECHO, and the time at which the packet was generated. For ECHOREPLY packets, we again record the id field. We also record the round-trip time, obtained by subtracting the time the ECHOREPLY was received from the time carried in the packet's payload. Since all timestamps are provided by a single host, synchronized clocks are not needed.

Our trace format supports measurements from a variety of network devices; however, this paper focuses exclusively on the AT&T WaveLAN packet radio device. This device operates in the 900MHz region, and offers a nominal bandwidth of 2 Mb/s. The static infrastructure for our WaveLAN network consists of a collection of base stations called WavePoints that serve as bridges to an Ethernet. A roaming protocol triggers handoffs between WavePoints as a WaveLAN host moves. The WaveLAN device reports signal characteristics such as signal level, signal quality and silence level, which we record along with packet traffic.

3.1.2 Collection Method

To collect traced information we have an in-kernel implementation, similar to other network data collection platforms [9, 11], that provides accurate timing with modest overheads. Hooks are placed in the input and output routines of traced devices to allow the tracing software access to packets. If tracing is enabled, the packet tracing routine examines the media header and encapsulated packet to ensure that the packet is one of the traced types. It then copies relevant information from the packet into a circular buffer. Periodically, the kernel examines the device performance parameters and places that information into the same buffer. Since the kernel buffer is limited in size, it may be overrun. In that case, we are careful to keep track of the number and type of lost records.

The kernel exports a pseudo-device supporting `open`, `close`, and `read` operations: opening the pseudo-device enables tracing; closing it disables tracing. A daemon process periodically extracts collected data from the pseudo-device and writes it to disk.

3.2 Distillation Phase

Trace distillation is the process of transforming a collected trace into a list of performance parameters for a simple network model. This list is called a *replay trace*, and each entry in it specifies latency, bandwidth, and loss rate for the duration of time indicated in that entry.

This section begins with a discussion of our network performance model, and then describes the algorithm which derives parameters for this model. It is important to note that the model is separable from the methodology. The use of a different model may, of course, require changes to details in all phases of the methodology.

3.2.1 A Simple, Instantaneous Network Model

Simplicity rather than sophistication is the keystone of our network model. Our goal is to use the simplest possible model capable of yielding accurate results in modulation. Much of the complexity of wireless networks arises from temporal variation caused by changes in location, physical environment, or cross traffic. We cope with this complexity while preserving simplicity by decomposing time-varying behavior into a sequence of short intervals of invariant behavior, much as a complex curve can be approximated by many short line segments.

We place three constraints on the model used in modulation. First, it must be possible to obtain the parameters of the model solely from observations at an endpoint. Second, the workload required by the model in the trace collection phase should not significantly perturb the network. Third, it must be cheap to compute the parameters, and to use them during modulation.

Single Packet Consider two hosts, H_1 and H_2, with H_1 sending packets to H_2 over some network N. This network may also be carrying cross traffic, defined as any traffic through N that is not between H_1 and H_2. Such cross traffic may change over time, and will affect the delays and losses experienced by any traffic between H_1 and H_2.

We can model the end-to-end path from H_1 to H_2 as a series of m service queues, $q_1, q_2, \ldots q_m$. We model each queue's instantaneous service time deterministically, as

$$t_k = f_k + sv_k \tag{1}$$

where t_k is the total delay imposed by queue q_k, f_k is a fixed, per-packet cost, s is the size of the packet in bytes, and v_k is a variable, per-byte cost. In physical terms, v_k is the inverse of the instantaneous bandwidth of the network element k; f_k is the current transmission latency of that element, and is the sum of queueing, per-packet processing, and propagation delays. For a single packet traversing this network, the total delay experienced from H_1 to H_2 is:

$$\begin{aligned} \Delta &= \sum f_k + s \sum v_k && (2) \\ &= F + sV && (3) \end{aligned}$$

It is important to note that each queue k services cross traffic as well as direct traffic between H_1 and H_2. This means that the delays and losses experienced by traffic between H_1 and H_2, and hence the values of f_k and v_k, change over time as cross-traffic load changes. The quantities f_k and v_k are thus intended to capture delays as experienced by traffic from host H_1 to H_2, rather than some static properties of the queue.

Multiple Packets In keeping with the constraints on our model, we take a simplistic approach to queueing delay when considering multiple packets from H_1 to H_2 through N. As above, an individual queue's service time is broken up into f_k, the transmission latency over that portion of the network, and v_k, the per-byte cost induced by bandwidth constraints. Our model holds that the maximum throughput at any individual queue is dependent only on the v_k term and the sizes of packets. Single single-byte latency, f_k, can be overlapped and causes no queuing delay.

Since the connection between hosts H_1 and H_2 is a serial string of such queues, the overall throughput is determined by the largest per-byte cost, $\max(v_k)$ at some particular q_k. We call this *bottleneck* queue q_b, its per-byte costs, V_b, and the residual per-byte costs, V_r. By definition, $V = V_b + V_r$, allowing us to rewrite Equation 3 as

$$\Delta = F + s(V_b + V_r) \tag{4}$$

For a given time segment of duration d, there are a single set of *delay parameters*: F, V_b, and V_r; we combine these into a *delay tuple* of the form $\langle d, F, V_b, V_r \rangle$. By composing a set of these delay tuples into a list, $\mathcal{D}$, we can model arbitrary changes in delay as perceived by traffic flowing from H_1 to H_2 over N.

However, delay is only one aspect of network performance; the other is packet loss. We model losses as a probability L of dropping a given packet during the interval d; each packet in d has the same chance of being dropped. Our model, which is above the datalink layer, assumes that corrupt packets are coerced to lost ones. We can combine instantaneous loss and delay behavior into a sequence of five-element *network quality tuples*, $\mathcal{S}$, of the form $\langle d, F, V_b, V_r, L\rangle$. Such a sequence, the replay trace, describes network quality over time.

3.2.2 Obtaining Model Parameters

Our strategy for producing a replay trace involves observing the loss and delay behavior of a known, fixed workload on the network. Since clock drift on our trace collection hardware can be significant relative to the time scale of network quality variation, we are forced to use a strategy that depends only on timestamps taken on a single host. This implies that the workload must consist of round-trips, and that we must assume that network delays are symmetric. These assumptions could be removed if our trace collection hosts were equipped with high-resolution, low-drift, synchronized clocks.

As mentioned in Section 3.1.1, the known workload is provided by a variant of the `ping` program. Our workload sends out a group of three packets each second, in two stages. In the first stage, `ping` sends an ECHO request with a small data payload of size s_1 to a target host. When the corresponding ECHOREPLY is received, `ping` begins the second stage by sending two larger ECHO requests of size s_2, back-to-back, to the same host. As described in Section 3.1.1, we record the time at which the ECHOREPLY packets arrive, the round-trip times, and the sizes of those packets. Since these three packets were sent very close together in time, we normally assume they observed the same network conditions; our technique detects and corrects for situations when this assumption does not hold.

Delay The production of one estimate of instantaneous network delay requires two steps. The first step is determining the end-to-end latency, F, and the total per-byte costs, V. The second step involves discovering the relative proportions of V_b and V_r. During these two steps, we implicitly observe and record losses.

The first packet takes some time t_1 for its round-trip, the second some longer time t_2. Since their round-trips were entirely non-overlapping, we know the second packet incurred no queueing delay due to the first packet. For these packets, each taking a round trip, our model says that:

$$t_1 = 2(F + s_1 V) \tag{5}$$
$$t_2 = 2(F + s_2 V) \tag{6}$$

From these equations we can determine F and V.

The second and third packets each have size s_2, but the second takes time t_2, and the third t_3, where $t_3 > t_2$. Since they were sent back-to-back, the third packet is subject to queueing delay behind the second. Hence our model says that:

$$t_2 = 2(F + s_2(V_b + V_r)) \tag{7}$$
$$t_3 = 2(F + s_2(V_b + V_r)) \quad + \quad s_2 V_b \tag{8}$$

That is, that the third packet is delayed behind the second at the bottleneck queue, and that delay is exactly $s_2 V_b$. Note that the bottleneck cost is paid only on the outbound leg; on the inbound leg, the third packet is already delayed far enough behind the second to get through the bottleneck queue without extra delay. These two equations, together with the first two, give us one set of F, V_b, V_r estimates.

Occasionally, solving these equations for a single group of packets results in a negative value for one or more of the parameters. Such values arise when the packets in the group experienced substantially different networking conditions from each other. In such situations, we plug in the immediately preceding observed parameters, and take the difference between the expected and observed times. We apply the difference to F and reuse the previous V_b and V_r, reasoning that short-term performance variation is most likely due to media access delay. We are careful not to let this corrective factor cascade.

We use a sliding-window algorithm to convert these estimates into components of an element of $\mathcal{D}$, a delay tuple. Each step produces an average of estimates in the current window. Our choice of window width, five seconds, balances the desire to discount outlying estimates with the need to be reactive to true change in network conditions.

Loss To estimate the loss rate, L, we examine sequence numbers of the ECHOREPLY packets in and immediately surrounding the current window. This tells us, for the time from the last packet before the window to the first packet after the window, how many ECHOREPLY packets were expected but failed to arrive. We know that we received b ECHOREPLY packets, but expected to receive a, the number of ECHO packets sent. Let $P = 1 - L$ be the unknown probability that a packet sent arrives without being dropped. Thus, we know that a ECHO packets were sent, of which Pa arrived at the target host. For each of those Pa packets, the target host responds with an ECHOREPLY, of which P^2a arrive back at the original sender. But, since we know we received b ECHOREPLY packets:

$$b = P^2 a \tag{9}$$
$$L = 1 - \sqrt{b/a} \tag{10}$$

Combining L and $\mathcal{D}$, we obtain a network quality tuple, which is an element of $\mathcal{S}$. The algorithm to produce the entire list runs in the order of the length of the trace, and comprises a single pass.

No clocks are used in calculating L, so in fact we could dispense with the symmetry assumption for loss information. However, to simplify the implementation, we preserve the more restrictive assumption of symmetry already needed for calculating delay parameters. While we have observed slightly asymmetric loss rates in our wireless devices, our assumption of symmetry does not significantly affect the accuracy of modulation.

3.3 Modulation Phase

Once we have a description of network behavior in a replay trace, we must reproduce that behavior in a wired networking environment. We do this through two components: a user-level daemon process and an in-kernel modulation layer. The daemon feeds lists of network quality tuples, $\mathcal{S}$, to the kernel, which subjects all inbound and outbound traffic to delays and drops according to the model.

When modulation begins, the daemon opens a pseudo-device much like that described in Section 3.1.2. The daemon then reads the quality tuples from a file, and writes them to the pseudo-device. This device is backed by an in-kernel, fixed-sized buffer. When the buffer is full, the daemon blocks until there is room to write new tuples. The daemon may write a file of tuples once and then close the pseudo-device, or it may loop over the file until interrupted.

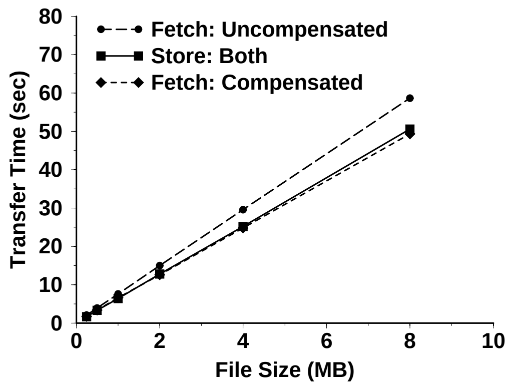
The modulation layer is placed between the IP and Ethernet layers of the protocol stack. This modified protocol stack is used by the host on which we wish to run experiments. This layer uses an on-line scheduling algorithm to delay and drop packets according to the parameters found in a replay trace.

The modulation layer reads out entries from this buffer as they are needed, and subjects packet traffic to these parameters by placing them in a delay/drop queue, where they are scheduled asynchronously. Given our linear model, delay calculations are straightforward. To ensure that outbound and inbound packets interfere with one another, we use a single delay queue. Outbound and inbound packets are scheduled exactly the same way by this queue. In addition to calculating a packet's delay, we generate a random number to decide whether or not to drop it. Lost packets are dropped only after they pass through the bottleneck queue.

Scheduling Granularity In our current implementation, clock-based interrupt resolution on our hosts is only 10 milliseconds. To cope with this limited resolution, we make the simplifying assumption that packet arrivals, and hence departures, are uniformly distributed between clock ticks. Hence, if we schedule on the closest clock tick, the long term average error should tend to zero. Because packets to be delayed less than half a clock tick are sent immediately, sparse traffic modeled over relatively high-performance links will not be sufficiently delayed. This simplifying assumption could be avoided in one of two ways. One approach would be to use a custom hardware clock, but this would preclude our ability to run on stock machines. The other approach, which we rejected in the interests of minimal system perturbation, would be to raise the frequency of clock interrupts as described by Ahn et al [1].

Delay Compensation Our intent is to provide symmetric delay of inbound and outbound traffic; that is, for a fixed set of modulation parameters, inbound traffic should perform exactly the same as outbound traffic. However, because the unified delay queue is placed at an endpoint, inbound and outbound delays are slightly asymmetric, as shown in Figure 1. In this figure, a synthetic trace roughly equivalent to that of a WaveLAN is used to modulate FTP transfers of varying sizes, both inbound and outbound. The uppermost, dashed curve in this figure shows the fetch FTP performance with this implementation; the solid line shows the store FTP performance. Without modifications, inbound traffic has significantly lower throughput than outbound, an artifact of the asymmetric placement of the delay queue. An accurate realization of our model would delay these two streams identically.

To correct for this, we compensate for the additional delays on inbound traffic. To determine the amount of compensation, we measure the physical network over which modulation will take place, using the same tools described here. This measurement need occur only once; it is independent of the network to be emulated. We then take the long-term average of the modulating network's bottleneck per-byte costs,



This figure depicts replay of a synthetic trace whose performance is close to that of a WaveLAN device, both with and without inbound traffic compensation. A perfect realization of our delay model would result in identical performance for Fetch and Store.

Figure 1: Effect of Delay Compensation

V_b. This term is subtracted from the replay trace's bottleneck per-byte costs for modulation of inbound packets

The effectiveness of compensation is shown in Figure 1. Store throughput does not change, but fetch throughput with compensation, shown by the dotted curve, is much closer to that of send. To validate the assertion that compensation depends only on the modulation setup, an experiment with a synthetic trace was run for a much slower network. The results confirm that compensation is independent of the traced network performance.

4 Validation Approach

Our goal in trace modulation is to subject a system to a networking environment indistinguishable from the one on which the trace was collected. To gauge our success in this, we collected and distilled traces from a small set of interesting mobile networking scenarios. We then compared the end-to-end performance of three benchmarks under trace modulation to live performance on those scenarios.

We chose WaveLAN because we depend on it heavily in day-to-day use, and understanding its detailed characteristics is important to our community. However, this choice is an especially stressful test case for our methodology because WaveLAN is a fast medium and packet delays are short. The accuracy of emulation is therefore likely to be particularly sensitive to limitations of our model and shortcomings in our implementation.

4.1 Mobile Scenarios and Traces

The four scenarios we have chosen for evaluation were conducted at Carnegie Mellon University, and were chosen to cover a wide range of user behavior and network quality. Figures 2 through 5 present key network characteristics of these scenarios.

The topmost component of each figure depicts the observed signal level in WaveLAN-specific units. Higher levels indicate stronger signals; levels below 5 are assumed to be background noise by the WaveLAN driver. The other three components of each figure depict quantities derived from the distilled traces: latency in milliseconds, bandwidth in kilobits per second, and loss rate in percent. To account for temporal variation, four trials were obtained of each scenario. Each graph combines the observations from all four trials.

In Figures 2 through 4, the X axis represents location, with labels indicating checkpoints along the path. Although every effort was made to keep physical speed identical across trials of a scenario, perfect consistency is impossible. To account for this, we normalized inter-checkpoint intervals to be of the same length across different trials of a given scenario; these lengths roughly correspond to the proportion of time that interval took with respect to the entire trace. At each X value, the vertical line represents the range of observed parameter values at that location across different trials. For example, at location x4 in the first graph of Figure 2, the minimum observed signal level was 17 and the maximum was 22.

Since the fourth scenario does not involve motion, it is meaningless to attempt to correlate parameter values with locations. Hence Figure 5 depicts the occurrence of observed values as histograms.

4.1.1 Porter: Inter-Building Travel

The *Porter* trace, depicted in Figure 2, begins in the main lobby of Wean Hall (location x0 in the graphs), then traverses an outdoor patio to Porter Hall (x1-x3), and finally enters and traverses Porter Hall (x4-x6).

Signal level is highly variable initially, but steadily improves as the Wean-Porter patio is crossed. It falls off again as Porter Hall is traversed. Close to location x5 in Porter Hall, signal level again becomes highly variable. The latency graph indicates several spikes as high as 100 milliseconds, but typically hovers between 1.5 and 10 milliseconds. The bandwidth graph shows typical rates between 1.4Mb/s and 1.6Mb/s, but also indicates spikes as low as 900Kb/s. Loss rates are typically below 10%, the worst cases being the early portion of the Wean-Porter patio, and the end of Porter Hall.

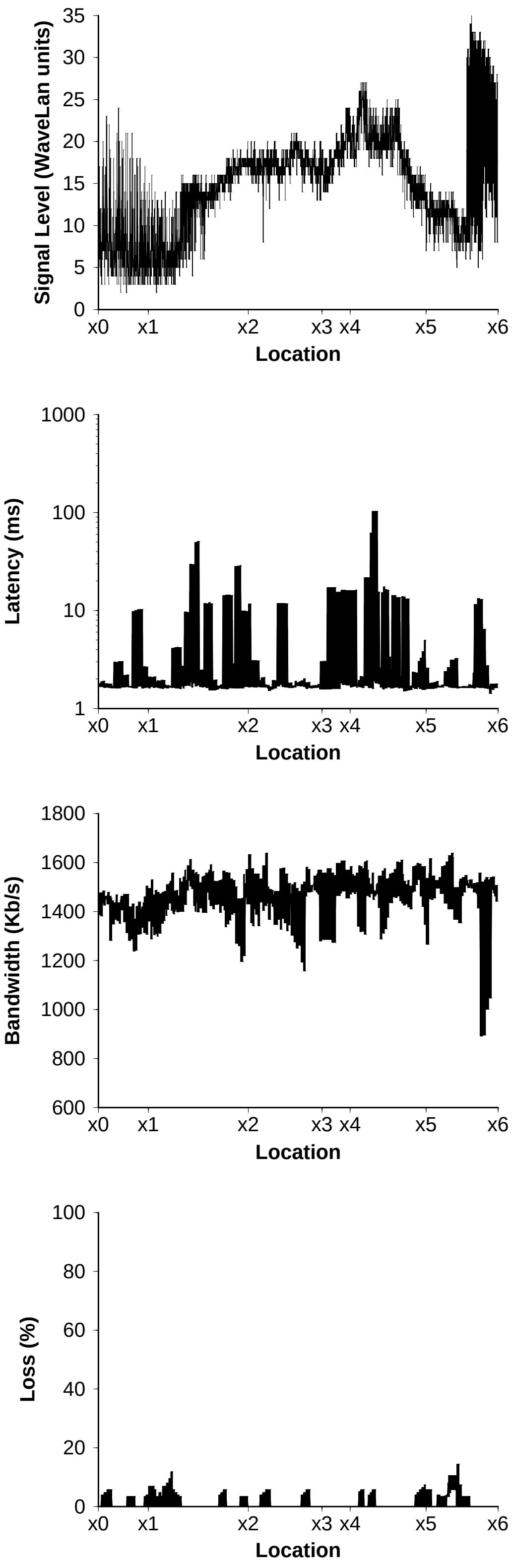
4.1.2 Flagstaff: Outdoor Travel

Flagstaff, the next scenario, is depicted in Figure 3. The path for this scenario leaves Porter Hall (y0-y1) to walk along the back edge of the campus in Schenley Park (y1-y5), then around Flagstaff Hill (y5-y9). The entire trace takes place outdoors, but at all times we remain in the line of sight of buildings housing WavePoint base stations.

Overall, signal quality during the Flagstaff traces is somewhat below that of the Porter traces. It starts off highly variable, then falls off sharply as soon as Schenley Park is entered, and stays roughly constant at a low level thereafter. On the whole, latency is much better in Flagstaff than in Porter. Average bandwidth is somewhat better in the Flagstaff traces than Porter. Where the Flagstaff traces are significantly worse than the Porter traces is in loss rate, particularly later in the traversal.

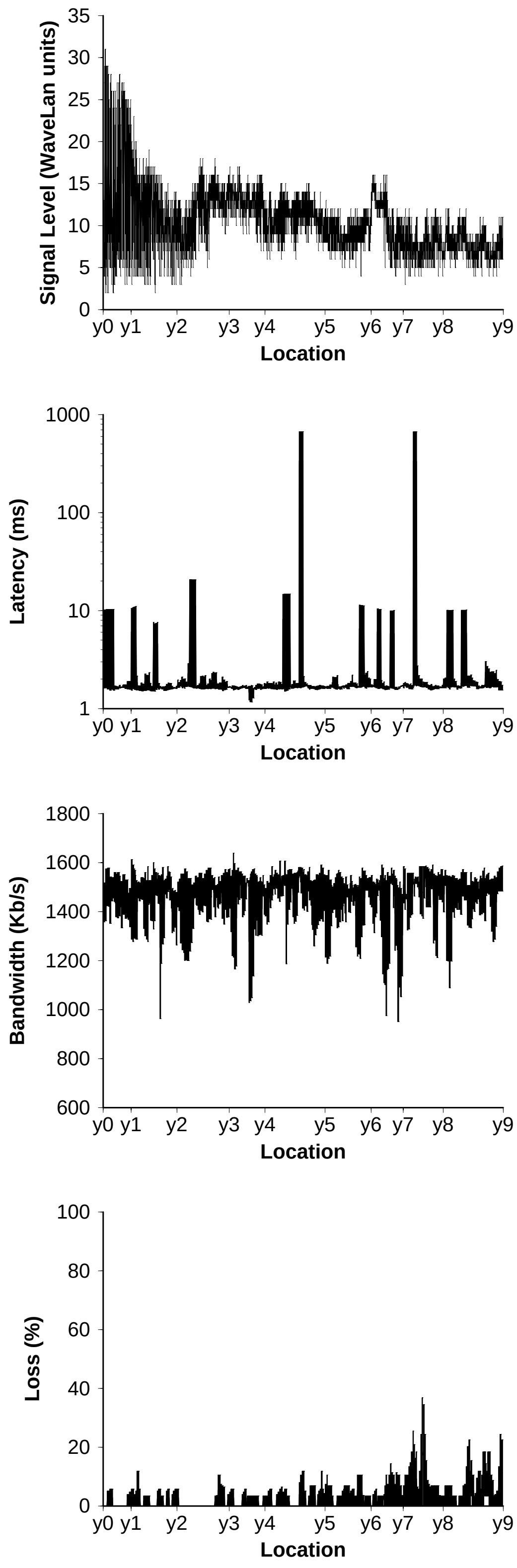
4.1.3 Wean: Traveling to Classroom

The next scenario is traveling from a graduate student office to a classroom, all within Wean Hall. This trace, depicted in Figure 4, is called the *Wean* trace. The trace begins in an office with known poor connectivity (z0), then traverses a hallway to the building's elevator (z0-z3). We wait for the



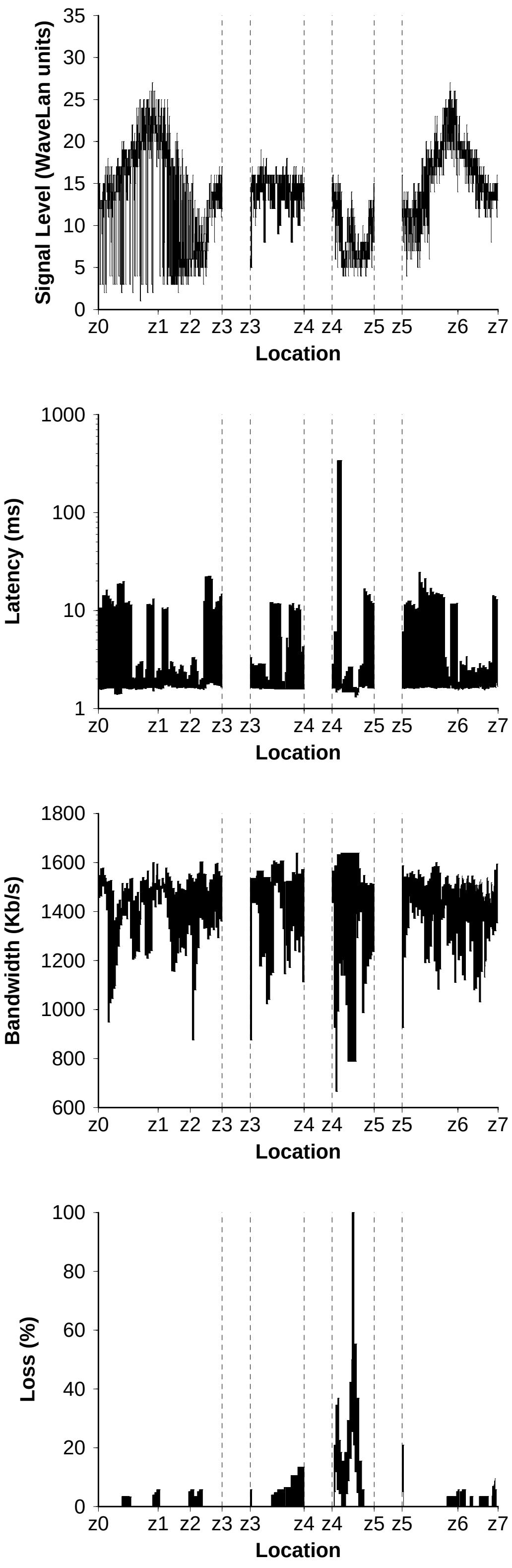
This figure shows observed signal quality and derived model parameters for the Porter scenario. At each X value, the vertical line gives the range of observations at that location across trials. Note the log scale for latency.

Figure 2: Porter Traces



This figure shows observed signal quality and derived model parameters for the Flagstaff scenario. At each X value, the vertical line gives the range of observations at that location across trials. Note the log scale for latency.

Figure 3: Flagstaff Traces



This figure shows observed signal quality and derived model parameters for the Wean scenario. At each X value, the vertical line gives the range of observations at that location across trials. Note the log scale for latency.

Figure 4: Wean Traces

elevator (z3-z4), then enter and ride it three floors (z4-z5). We then exit the elevator and walk to the classroom (z5-z7). Since this scenario involves discontinuous motion, the graphs in this figure are broken into four regions: the walk to the elevator, the wait for the elevator, riding the elevator, and the walk to the classroom.

Signal level is variable, but acceptable for the entire walk to the elevator. While waiting, signal level is quite good, but on the elevator ride it drops precipitously. On exiting the elevator, signal level is again good during the walk to the classroom. Latency is good except during the elevator ride, peaking at 350 milliseconds. Bandwidth is somewhat lower than that found in the Porter traces. Loss rates are low except for the duration of the elevator ride, where they are atrocious.

4.1.4 Chatterbox: Busy Conference Room

The final scenario is intended to capture the effect of interfering wireless traffic rather than physical motion. The trace collection host is placed in a room with five other laptops also using WaveLAN. Each of the other laptops continuously executes a workload produced by SynRGen [5], a synthetic file reference generator. The synthetic workload models a user in a edit-debug cycle on files stored on a remote NFS [15] file server.

We refer to this scenario as the *Chatterbox* scenario, depicted in Figure 5. This figure differs from the depictions of the previous three scenarios because there is no physical motion. We use simple histograms rather than plotting parameter values along a sequence of checkpoint locations. The difference in depiction limits us to coarse comparisons to the previous scenarios.

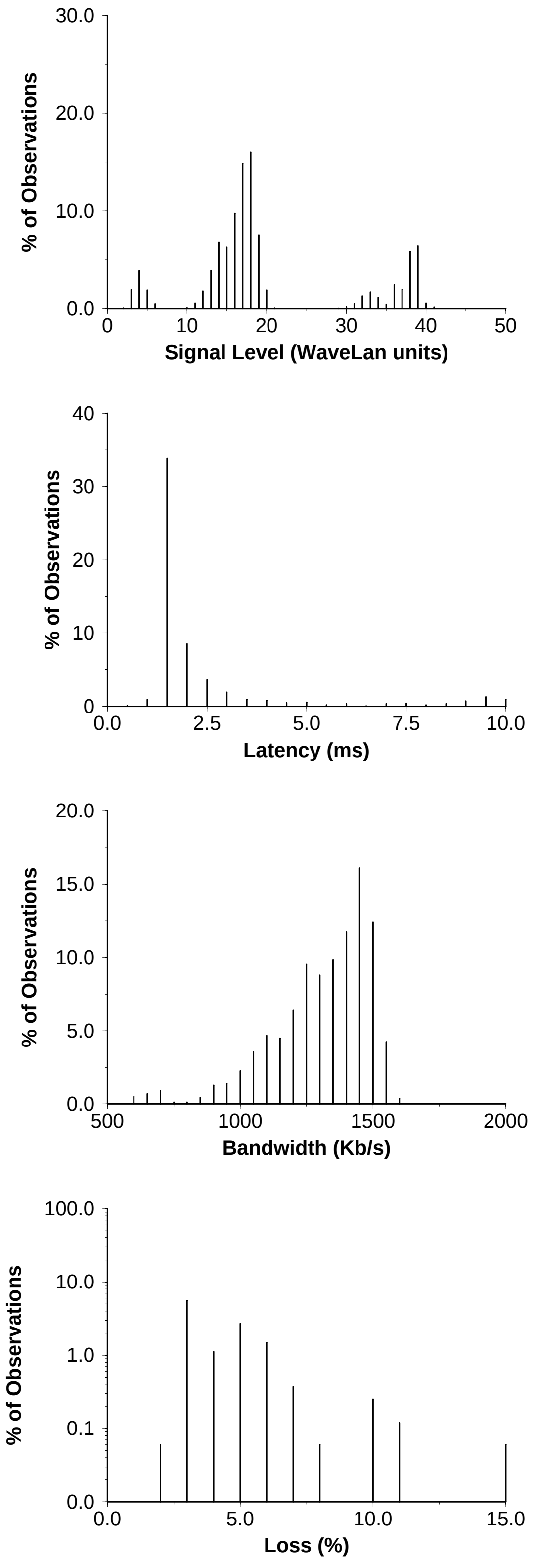
Figure 5 shows that signal level is consistently high, typically around 18. In spite of high signal level, the presence of interfering traffic results in poorer latency and bandwidth relative to previous scenarios. Loss rates are reasonable.

4.2 Benchmarks

We have selected three benchmarks to test the accuracy of trace modulation. The first benchmark involves a World Wide-Web workload [17]. Web reference traces of five users performing search tasks are replayed as fast as possible on a modified Mosaic v2.6 browser. To ensure good experimental control, all objects referenced in these traces were copied to a private Web server, and all URLs in the Web traces were changed to refer to this server.

The second benchmark is FTP. In this benchmark, we transfer a single 10MB file disk-to-disk, both to and from a laptop. This benchmark makes heavy use of the wireless network, and is designed to highlight any potential asymmetry in WaveLAN performance. This is especially important given the assumption of network symmetry forced upon us by the lack of synchronized clocks during trace collection, as discussed in Section 3.2.2.

The third benchmark is the Andrew benchmark run on NFS, a commonly-used network file system. Since NFS was not designed for a mobile environment, it makes no special attempt to defer or eliminate traffic on networks of low quality. We are careful to flush the NFS cache before each trial of the experiment. This benchmark contains an interesting mix of network traffic, and uses UDP as its transport protocol; in contrast, both FTP and HTTP use TCP as their transport protocol.



This figure shows observed signal quality and derived model parameters for the Chatterbox scenario. Unlike previous scenarios, there is no physical movement. Thus, all graphs depict distributions of observed values. Note the log scale for loss rate.

Figure 5: Chatterbox Traces

5 Experimental Results

5.1 Experimental Conditions

Trace collection and benchmarking were performed on an IBM ThinkPad 701c laptop with an Intel 80486 75MHz DX4 processor and 24MB of memory. We used a WaveLAN radio modem on the laptop, and relied on an infrastructure of several dozen WavePoint base stations to provide service to the mobile host through a single IP router. The laptop communicated with an Intel Pentium 90MHz workstation with 32MB of memory connected to the campus network via Ethernet. For modulation experiments, we used these same machines, but connected them with an isolated Ethernet. Both machines ran NetBSD version 1.2, customized for trace collection and modulation. In our experiments, only the ThinkPad performed collection or modulation.

For each benchmark on each scenario, we ran four live trials, and collected four traces for distillation, interleaving trials with traces. We then distilled each of the four traces for use in modulation, and ran a trial of the benchmark over each of these distilled traces. As one might expect, two trials of the same benchmark over the same distilled trace show little variance. When the same benchmark is run over distinct distilled traces intended to duplicate the same path, the results can show significant variance.

5.2 World Wide Web Benchmark

Figure 6 presents the results from the World Wide Web benchmark. In all scenarios, the difference between the means of real and modulated elapsed times is less than the sum of their standard deviations. This indicates that trace modulation is accurate within the bounds of experimental error for these scenarios.

5.3 FTP Benchmark

The FTP benchmark is important for two reasons. First, it is network-limited, and therefore most sensitive to network performance. Second, since send and receive performance are largely independent, it allows us to explore the impact of our network symmetry assumption forced by the lack of synchronized clocks during trace collection.

Figure 7 presents our results. In the Wean and Chatterbox scenarios, real and modulated performance are comparable: the difference between the means is less than the sum of their standard deviations. While this is not true of the Flagstaff scenario, the real send and receive performance differ by more than 20 seconds. Both modulated send and receive performance are very close to the mean of real send and receive, and hence an accurate recapturing of the traced environment given our symmetry limitations. The Porter scenario is the only troubling one, not sufficiently delaying either send or receive traffic. Modulated send performance is off by 1.05 times the sum of the standard deviations; receive is off by 1.56 times.

The substantial difference between send and receive performance over the real WaveLAN in these scenarios indicates that network performance is in fact quite asymmetric. This contradicts the modeling assumption of symmetry stated in Section 3.2.2, and further emphasizes the need for synchronized clocks during trace collection; such clocks would allow us to trace one-way performance. Also of note is the large standard deviation in the Chatterbox scenario. This high variance is shown in almost all of our real and modulated results.

Scenario	Real (s)		Modulated (s)	
Wean	161.47	(7.82)	160.04	(2.60)
Porter	159.83	(5.07)	150.65	(5.83)
Flagstaff	157.82	(6.58)	148.64	(9.61)
Chatterbox	169.07	(17.63)	157.62	(10.18)
Ethernet	140.30	(3.07)	—	—

This table gives the mean elapsed time in seconds of four trials of the World Wide Web benchmark for each mobile scenario. For reference, the last row gives the performance of the benchmark over the Ethernet used for modulation. Figures in parentheses are standard deviations. Note that due to a problem with our experimental setup, the real Porter numbers come from only three trials rather than four.

Figure 6: Elapsed Times for World Wide Web Benchmark

Scenario		Real (s)		Modulated (s)	
Wean	send	79.88	(10.88)	72.65	(3.33)
	recv	64.93	(0.93)	67.83	(2.34)
Porter	send	86.38	(4.94)	76.65	(4.29)
	recv	82.23	(1.92)	72.95	(4.01)
Flagstaff	send	88.15	(1.60)	74.88	(2.97)
	recv	61.85	(1.12)	70.80	(3.36)
Chatterbox	send	116.83	(30.49)	92.13	(20.13)
	recv	96.83	(42.15)	87.28	(17.18)
Ethernet	send	20.50	(0.08)	—	
	recv	18.83	(0.17)		

This table gives the mean elapsed times in seconds of four trials of the FTP benchmark. Send and receive performance are reported separately. For reference, the final row gives benchmark performance over the Ethernet used for modulation. Numbers in parentheses are standard deviations.

Figure 7: Elapsed Times for FTP Benchmark

5.4 Andrew Benchmark on NFS

The input to the Andrew Benchmark is a tree of about 70 source files occupying about 200KB. There are five distinct phases in the benchmark: MakeDir, Copy, ScanDir, ReadAll, and Make. The benchmark is run over files stored in NFS. Roughly, there are two classes of NFS operations: status checks and data exchanges. The former are typically very small messages, while the latter are larger. For most NFS clients, the ScanDir and ReadAll phases operate on warm caches, and transmit only status-check messages.

Figure 8 presents the elapsed times for each phase under real and modulated network conditions, as well as the total times for the benchmark. In three of the four scenarios — Wean, Porter, and Chatterbox — the difference between the means of real and modulated total times is within the sum of their standard deviations. In two of those three, Porter and Chatterbox, this is also true of all individual phases of the benchmark.

In the Wean trace, the ScanDir and ReadAll phases are both under-delayed in modulation. We suspect this is due to our inability to schedule delays at granularities shorter than 10 milliseconds. Many of the short, infrequent messages exchanged during those two phases do not have calculated delays large enough to be acted upon.

We conjecture that the Flagstaff scenario, where real and modulated performance diverge the most, also suffers from this phenomenon. As shown in Figures 2 through 5, Flagstaff latency and bandwidth tend to be comparable to

Scenario		MakeDir (s)	Copy (s)	ScanDir (s)	ReadAll (s)	Make (s)	Total (s)
Wean	Real	3.00 (0.00)	18.00 (0.82)	13.50 (0.58)	23.00 (0.82)	105.50 (3.87)	163.00 (4.40)
	Mod.	2.50 (1.00)	19.25 (3.69)	10.00 (1.83)	19.00 (0.82)	112.00 (4.97)	162.75 (4.86)
Porter	Real	3.00 (0.00)	20.00 (1.41)	18.50 (4.65)	23.50 (1.29)	104.50 (1.29)	169.50 (5.45)
	Mod.	2.50 (1.00)	16.75 (3.86)	12.00 (4.24)	20.25 (2.87)	99.50 (4.20)	151.00 (14.09)
Flagstaff	Real	2.75 (0.50)	19.25 (0.50)	15.25 (1.26)	28.00 (1.83)	111.75 (2.99)	177.00 (4.69)
	Mod.	2.75 (0.50)	15.00 (2.71)	10.75 (3.59)	20.00 (2.83)	97.25 (4.92)	145.75 (5.91)
Chatterbox	Real	3.50 (1.00)	22.50 (5.69)	18.25 (3.96)	27.25 (6.55)	109.25 (11.18)	180.75 (27.61)
	Mod.	4.00 (0.82)	30.75 (19.43)	21.00 (18.92)	30.00 (12.44)	117.00 (20.61)	202.75 (50.79)
Ethernet	Real	2.25 (0.50)	12.50 (0.58)	7.75 (0.50)	17.50 (0.58)	84.00 (1.41)	124.00 (1.63)

This table gives the per-phase mean elapsed times in seconds, and the total benchmark time, of four trials of the Andrew Benchmark under real and modulated network conditions. For reference, the final row gives benchmark performance over the Ethernet used for modulation. Standard deviations are given in parentheses.

Figure 8: Elapsed Times for Andrew Benchmark Phases

or better than those of the other three scenarios. Thus, many of the shorter NFS messages that are present in the benchmark will have delays below our threshold.

5.5 Discussion

As our benchmarks show, trace modulation is an effective way to provide reproducible yet realistic network performance in a controlled experimental environment. While the technique works quite well, there are some avenues for further investigation. As the FTP performance over WaveLAN clearly shows, we have need for fine-grained, low-drift, synchronized clocks to accurately model asymmetric network behavior. As the Andrew Benchmark shows, another use for such clocks would be to schedule delayed packets more accurately than the current NetBSD kernel allows.

While Chatterbox results show that we adequately capture a congested environment, we have some reservations with our approach; the variance observed during our benchmarks is high enough to warrant further study. Such variance may be from several sources such as the bursty behavior of our SynRGen synthetic users, and media contention. Our current approach of including extra delay encountered by one or more packets in a triplet as extra latency performs reasonably well, but we suspect a more complicated model will be required to fully capture the behavior exhibited in this scenario.

6 Conclusion

In this paper, we have shown how the characteristics of a complex network can be captured and reproduced. Although motivated by the demands of wireless networks, this approach also holds promise for wired networks given a sufficiently fast network on which to perform modulation.

The essence of our strategy is a trace-based approach involving three phases. The first phase records observations of the wireless network when it is subjected to a known workload from a moving host. The second phase interprets these observations in the light of a simple network model, and transforms them into a replay trace. The third phase uses the replay trace to modulate the loss and delay behavior of a LAN attached to a static host. During modulation, unmodified application and system software on the static host perceive time-varying network quality faithful to that seen by the moving host during the first phase.

Our validation experiments confirm the effectiveness of this approach, but point out a need for fine-grain, low-drift, synchronized clocks. Such clocks would enable us to eliminate our assumption of network symmetry and hence allow us to use one-way rather than round-trip measurements. They would also allow us to accurately reproduce short delays.

While this paper has focused on the use of traces from real networks, it is also possible to modulate using *synthetic traces*. These can be used to generate characteristics that can only be approximated by actual networks. For example, a recent paper [14] reports on the use of synthetic traces to explore the behavior of an adaptive mobile system in response to step and impulse variations in bandwidth.

We are confident that the trace-based methodology described here will prove to be useful and versatile. Analyses of traces can offer broad design insights for mobile systems and help in choosing system parameter values. Tracing can play an important role in debugging by deterministically reproducing the network conditions under which a subtle bug was originally uncovered. Traces exported from demanding environments enable a new mobile system to be stress-tested under real-life conditions before it is ready for deployment. A set of traces can be used as a benchmark family for evaluating and comparing the adaptive capabilities of alternative mobile system designs. Overall, the strength of our methodology lies in its ability to combine realism with reproducibility — an asset that will prove invaluable in many aspects of mobile system design, implementation and evaluation.

Acknowledgements

The authors wish to thank Kevin Walker, who designed and generated the trace visualization graphs in Section 4.1, and Jason Flinn, who pointed out an earlier error in our probability of loss calculation. Thanks are also due to Dave Johnson, Bruce Maggs, and Hui Zhang who helped refine early versions of both our work and our description of it. Mary Baker, Mike Davis, Barbara Denny, Chane Fullmer, Geoff Kuenning, and Mark Lewis provided early feedback on our trace collection format and associated tools and documentation.

Software Availability

The software described in this paper is available for unrestricted use. Directions on obtaining source code can be found on the World Wide Web at `http://www.cs.cmu.edu /afs/cs.cmu.edu/project/coda/Web/coda.html`

References

[1] Ahn, J. S., Danzig, P. B., Liu, Z., and Yan, L. Evaluation of TCP Vegas: Emulation and Experiment. *Computer Communication Review 25*, 4 (August 1995), 185–195.

[2] AT&T Global Information Solutions Company. Architecture Specification for WaveLAN Air Interface.

[3] Davies, N., Blair, G. S., Cheverst, K., and Friday, A. A Network Emulator to Support the Development of Adaptive Applications. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location Independent Computing* (April 10-11 1995).

[4] Dawson, S., and Jahanian, F. Probing and Fault Injection of Dependable Distributed Protocols. *The Computer Jouranl 38*, 4 (1995).

[5] Ebling, M., and Satyanarayanan, M. SynRGen: An Extensible File Reference Generator. In *Proceedings of the 1994 ACM SIGMETRICS Conference* (Nashville, TN, May 1994).

[6] Eckhardt, D., and Steenkiste, P. Measurement and Analysis of the Error Characteristics of an In-Building Wireless Network. *Computer Communication Review 26*, 4 (October 1996), 243–254.

[7] Howard, J. H., Kazar, M. L., Menees, S. G., Nichols, D. A., Satyanarayanan, M., Sidebotham, R. N., and West, M. J. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems 6*, 1 (February 1988).

[8] Ingham, D. B., and Parrington, G. D. Delayline: A Wide-Area Network Emulation Tool. *Computing Systems 7*, 3 (1994).

[9] Jacobson, V., Leres, C., and McCanne, S. *The Tcpdump Manual Page.* Lawrence Berkeley Laboratory, Berkeley, CA.

[10] Keshav, S. Packet-Pair Flow Control. To appear in IEEE/ACM Transactions on Networking.

[11] McCanne, S., and Jacobson, V. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of the 1993 Winter USENIX Technical Conference* (San Deigo, CA, January 1993).

[12] Nguyen, G., Katz, R., Noble, B., and Satyanarayanan, M. A Trace-Based Approach for Modelling Wireless Channel Behavior. In *Proceedings of the Winter Simulation Conference* (1996).

[13] Noble, B., Nguyen, G., Satyanarayanan, M., and Katz, R. Mobile Network Tracing. Internet RFC 2041, October 1996.

[14] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles* (St. Malo, France, October 1997).

[15] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B. Design and Implementation of the Sun Network Filesystem. In *Usenix Conference Proceedings* (Summer 1985).

[16] Satyanarayanan, M. *RPC2 User Guide and Reference Manual.* School of Computer Science, Carnegie Mellon University, October 1991.

[17] Steere, D. C. *Using Dynamic Sets to Reduce the Aggregate Latency of Data Access.* PhD thesis, Carnegie Mellon University, School of Computer Science, 1996.