Network Working Group
Request for Comments: 2041
Category: Informational

B. Noble
Carnegie Mellon University
G. Nguyen
University of California, Berkeley
M. Satyanarayanan
Carnegie Mellon University
R. Katz
University of California, Berkeley

October 1996

Mobile Network Tracing

Status of this Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

Mobile networks are both poorly understood and difficult to experiment with. This RFC argues that mobile network tracing provides both tools to improve our understanding of wireless channels, as well as to build realistic, repeatable testbeds for mobile software and systems. The RFC is a status report on our work tracing mobile networks. Our goal is to begin discussion on a standard format for mobile network tracing as well as a testbed for mobile systems research. We present our format for collecting mobile network traces, and tools to produce from such traces analytical models of mobile network behavior.

We also describe a set of tools to provide network modulation based on collected traces. Modulation allows the emulation of wireless channel latency, bandwidth, loss, and error rates on private, wired networks. This allows system designers to test systems in a realistic yet repeatable manner.

1. Introduction

How does one accurately capture and reproduce the observed behavior of a network? This is an especially challenging problem in mobile computing because the network quality experienced by a mobile host can vary dramatically over time and space. Neither long-term average measures nor simple analytical models can capture the variations in bandwidth, latency, and signal degradation observed by such a host. In this RFC, we describe a solution based on network tracing. Our solution consists of two phases: trace recording and trace modulation.

In the trace recording phase, an experimenter with an instrumented mobile host physically traverses a path of interest to him. During the traversal, packets from a known workload are generated from a static host. The mobile host records observations of both packets received from the known workload as well as the device characteristics during the workload. At the end of the traversal, the list of observations represents an accurate trace of the observed network behavior for this traversal. By performing multiple traversals of the same path, and by using different workloads, one can obtain a trace family that collectively characterizes network quality on that path.

In the trace modulation phase, mobile system and application software is subjected to the network behavior observed in a recorded trace. The mobile software is run on a LAN-attached host whose kernel is modified to read a file containing the trace (possibly postprocessed for efficiency,) and to delay, drop or otherwise degrade packets in accordance with the behavior described by the trace. The mobile software thus experiences network quality indistinguishable from that recorded in the trace. It is important to note that trace modulation is fully transparent to mobile software --- no source or binary changes have to be made.

Trace-based approaches have proved to be of great value in areas such as file system design [2, 10, 11] and computer architecture. [1, 5, 13] Similarly, we anticipate that network tracing will prove valuable in many aspects of mobile system design and implementation. For example, detailed analyses of traces can provide insights into the behavior of mobile networks and validate predictive models. As another example, it can play an important role in stress testing and debugging by providing the opportunity to reproduce the network conditions under which a bug was originally uncovered. As a third example, it enables a system under development to be subjected to network conditions observed in distant real-life environments. As a final example, a set of traces can be used as a benchmark family for evaluating and comparing the adaptive capabilities of alternative

mobile system designs.

Our goal in writing this RFC is to encourage the development of a widely-accepted standard format for network traces. Such standardization will allow traces to be easily shared. It will also foster the development and widespread use of trace-based benchmarks. While wireless mobile networks are the primary motivation for this work, we have made every effort to ensure that our work is applicable to other types of networks. For example, the trace format and some of the tools may be valuable in analyzing and modeling ATM networks.

The rest of this RFC is organized as follows. We begin by examining the properties of wireless networks and substantiating the claim that it is difficult to model such networks. Next, in Section 3, we describe the factors that should be taken into account in designing a trace format. We present the details of a proposed trace format standard in Section 4. Section 5 presents a set of tools that we have built for the collection, analysis and replay of traces. Finally, we conclude with a discussion of related and future work.

2. Modeling Wireless Networks

Wireless channels are particularly complex to model, because of their inherent dependence on the physical properties of radio waves (such as reflections from "hard" surfaces, diffraction around corners, and scattering caused by small objects) and the site specific geometries in which the channel is formed. They are usually modeled as a time- and distance-varying signal strength, capturing the statistical nature of the interaction among reflected radio waves. The signal strength can vary by several orders of magnitude (+ or - 20-30 dB) within a short distance. While there have been many efforts to obtain general models of radio propagation inside buildings and over the wide area, these efforts have yielded inherently inaccurate models that can vary from actual measurements by an order of magnitude or more.

Signal-to-noise ratio, or SNR, is a measure of the received signal quality. If the SNR is too low, the received signal will not be detected at the receiver, yielding bit errors and packet losses. But SNR is not the only effect that can lead to losses. Another is inter-symbol interference caused by delay spread, that is, the delayed arrival of an earlier transmitted symbol that took a circuitous propagation path to arrive at the receiver, thereby (partially) canceling out the current symbol. Yet another problem is doppler shift, which causes frequency shifts in the arrived signal due to relative velocities of the transmitter and the receiver, thereby complicating the successful reception of the signal. If coherent reception is being used, receiver synchronization can be

lost.

More empirically, it has been observed that wireless channels adhere to a two state error model. In other words, channels are usually well behaved but occasionally go into a bad state in which many burst errors occur within a small time interval.

Developers of network protocols and mobility algorithms must experiment with realistic channel parameters. It is highly desirable that the wireless network be modeled in a thoroughly reproducible fashion. This would allow an algorithm and its variations to be evaluated in a controlled and repeatable way. Yet the above discussion makes it clear that whether analytical models are used or even actual experimentation with the network itself, the results will be either inaccurate or unlikely to be reproducible. A trace-based approach alleviates these problems.

3. Desirable Trace Format Properties

In designing our trace format, we have been guided by three principles. First, the format should be extensible. Second, it should be self-describing. Third, traces should be easy to manage. This section describes how each of these principles has affected our design.

Although we have found several interesting uses for network traces, it is certain that more will evolve over time. As the traces are used in new ways, it may be necessary to add new data to the trace format. Rather than force the trace format to be redesigned, we have structured the format to be extensible. There is a built-in mechanism to add to the kinds of data that can be recorded in network traces.

This extensibility is of little use if the tool set needs to change as the trace format is extended. Recognizing this, we have made the format -- particularly the extensible portions -- self-describing. Thus, old versions of tools can continue to work with extended traces, if perhaps in a less than optimal way.

In our experience with other tracing systems, management of trace files is often difficult at best. Common problems include the need to manage multiple trace files as a unit, not easily being able to extract the salient features of large trace files, and having to use dedicated trace management tools to perform even the simplest tasks. To help cope with file management, we have designed the the traces to be split or merged easily. To reduce dependence on specialized tools, we’ve chosen to store some descriptive information as ASCII strings, allowing minimal access to the standard UNIX tool suite.

4. Trace Format

This section describes the format for network traces. We begin by presenting the basic abstractions that are key to the trace format: the record, and the track, a collection of related records. We then describe the records at the beginning and end of a trace, the header and footer. The bulk of the section describes the three kinds of record tracks: packet, device, and general. These also make up the bulk of the actual trace. We conclude the section with a discussion of two special purpose records: the annotation and the trace data loss records.

4.1. Basic Abstractions

4.1.1. Records

A record is the smallest unit of trace data. There are several different types of records, each of which is discussed in Sections 4.2 through 4.7. All of the records share several features in common; these features are described here.

Records are composed of fields, which are stored in network order. Most of the fields in our records are word-sized. Although this may be wasteful in space, we chose to leave room to grow and keep trace management simple.

The first field in each record is a magic word, a random 32 bit pattern that both identifies the record's type and lends some confidence that the record is well formed. Many record types have both required and optional fields; thus they can be of variable size. We place every record's size in its second field. By comparing the size of a record to the known constraints for the record's type, we can gain further confidence that a record is well-formed. This basic record structure is illustrated in Figure 1.

All records also contain a two-word timestamp. This timestamp can take one of two formats: timeval or timespec. Only one of the two formats is used in any given trace, and the format is specified at the start of a trace file. The first word in either format is the number of seconds that have elapsed since midnight, January 1, 1970. The second word is the additional fractions of a second. In the timeval format, these fractions are expressed in microseconds, in the same way that many current operating systems express time. In the timespec format, these fractions are expressed in nanoseconds, the POSIX time standard. We've chosen these two values since they are convenient, cover most current and anticipated systems' notions of time, and offer appropriate granularity for measuring network events.

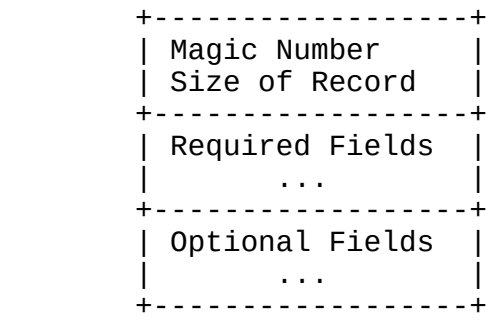


Figure 1: Record format

4.1.2. Tracks

Many of the record types have both fixed, required fields, as well as a set of optional fields. It is these options that provide extensibility to our trace format. However, to provide a self-describing trace, we need some compact way of determining which optional fields are present in a given record. To do this, we group related sets of packets into tracks. For example, a set of records that captured packet activity for a single protocol between two machines might be put together into a track. A track is a header followed by some number of related records; the header completely describes the format of the individual records. Records from separate tracks can be interleaved with one another, so long as the header for each individual track appears before any of the track's records. Figure 2 shows an example of how records from different tracks might be interleaved.

Track headers describe their records' content through property lists. An entry in a property list is a two-element tuple consisting of a name and a value. The name is a word which identifies the property defined by this entry. Some of these properties are measured only once for a track, for example, the address of a one-hop router in a track recording packets from that router. Others are measured once per record in that track, such as the signal strength of a device which changes over time. The former, which we call header-only properties, have their most significant name bit set. The value field of a header-only property holds the measured value of the property. Otherwise, the value field holds the number of words used in each of the track's records.

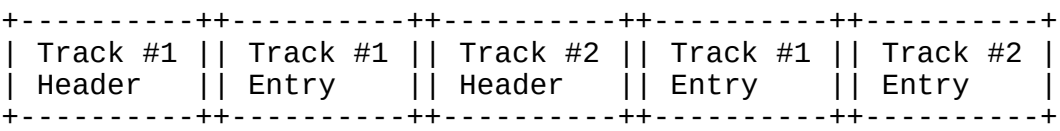


Figure 2: Interleaved track records

Those properties measured in each record in the track are grouped together in a value list at the end of each such record. They appear in the same order that was specified in the track header's property list so that tools can properly attribute data. Thus, even if a tool doesn't know what property a particular name represents, it can identify which parts of a trace record are measuring that property, and ignore them.

4.2. Trace Headers and Footers

Trace files begin with a trace header, and end with a trace footer. The formats of these appear in Figure 3. The header specifies whether this trace was collected on a single machine, or was merged from several other traces. In the former case, the IP address and host name of the machine are recorded. In the latter, the IP address is taken from the family of Class E address, which are invalid. We use a family of invalid addresses so that even if we cannot identify a number of hosts participating in the trace we can still distinguish records from distinct hosts.

```
#define TR_DATESZ   32
#define TR_NAMESZ   64

struct tr_header_t {
    u_int32_t        h_magic;
    u_int32_t        h_size;
    u_int32_t        h_time_fmt;         /* usec or nsec */
    struct tr_time_t h_ts;               /* starting time */
    char             h_date[TR_DATESZ];  /* Date collected */
    char             h_agent[TR_NAMESZ]; /* DNS name */
    u_int32_t        h_agent_ip;
    char             h_desc[0];          /* variable size */
};

struct tr_end_t {
    u_int32_t         e_magic;
    u_int32_t         e_size;
    struct tr_time_t  e_ts;        /* end time */
    char              e_date[32];  /* Date end written */
};
```

Figure 3: Trace header and footer records

The trace header also specifies which time stamp format is used in the trace, and the time at which the trace begins. There is a variable-length description that is a string meant to provide details of how the trace was collected. The trace footer contains only the time at which the trace ended; it serves primarily as a marker to show the trace is complete.

Unlike other kinds of records in the trace format, the header and footer records have several ASCII fields. This is to allow standard utilities some access to the contents of the trace, without resorting to specialized tools.

4.3. Packet Tracks

Measuring packet activity is the main focus of the network tracing project. Packet activity is recorded in tracks, with a packet header and a set of packet entries. A single track is meant to capture the activity of a single protocol, traffic from a single router, or some other subset of the total traffic seen by a machine. The required portions of packet headers and entries are presented in Figure 4.

Packet track headers identify which host generated the trace records for that track, as well as the time at which the track began. It records the device on which these packets are received or sent, and the protocol used to ship the packet; these allow interpretation of device-specific or protocol-specific options. The header concludes with the property list for the track.

```
struct tr_pkt_hdr_t {
    u_int32_t            ph_magic;
    u_int32_t            ph_size;
    u_int32_t            ph_defines;  /* magic number defined */
    struct tr_time_t     ph_ts;
    u_int32_t            ph_ip;       /* host generating stream */
    u_int32_t            ph_dev_type; /* device collected from */
    u_int32_t            ph_protocol; /* protocol */
    struct tr_prop_lst_t ph_plist[0]; /* variable size */
};

struct tr_pkt_ent_t {
    u_int32_t        pe_magic;
    u_int32_t        pe_size;
    struct tr_time_t pe_ts;
    u_int32_t        pe_psize;    /* packet size */
    u_int32_t        pe_vlist[0]; /* variable size */
};
```

Figure 4: Packet header and entry records

A packet entry is generated for every traced packet. It contains the size of the traced packet, the time at which the packet was sent or received, and the list of property measurements as specified in the track header.

The options we have defined to date are in Table 1. Several of these have played an important role in our early experiments. ADDR_PEER identifies the senders of traffic during the experiment. We can determine network performance using either PKT_SENTTIME for one-way traffic between two hosts with closely synchronized clocks, or round

trip ICMP ECHO traffic and the ICMP_PINGTIME option. Tracking PKT_SEQUENCE numbers sheds light on both loss rates and patterns. Section 5 discusses how these measurements are used.

4.4. Device Tracks

Our trace format records details of the devices which carry network traffic. To date, we've found this most useful for correlating lost packets with various signal parameters provided by wireless devices. The required portions of device header and entry records appear in Figure 5, and are quite simple. Device track headers identify the host generating the track's records, the time at which the observation starts, and the type of device that is being traced. Each entry contains the time of the observation, and the list of optional characteristics.

ADDR_PEER	Address of peer host
ADDR_LINK	Address of one-hop router
BS_LOC_X	One-hop router's X coordinate (header only)
BS_LOC_Y	One-hop router's Y coordinate (header only)
PKT_SEQUENCE	Sequence number of packet
PKT_SENTTIME	Time packet was sent
PKT_HOPS	Number of hops packet took
SOCK_PORTS	Sending and receiving ports
IP_PROTO	Protocol number of an IP packet
ICMP_PINGTIME	Roundtrip time of an ICMP ECHO/REPLY pair
ICMP_KIND	Type and code of an ICMP packet
ICMP_ID	The id field of an ICMP packet
PROTO_FLAGS	Protocol-specific flags
PROTO_ERRLIST	Protocol-specific status/error words

Table 1: Current optional fields for packet entries

```
struct tr_dev_hdr_t {
    u_int32_t            dh_magic;
    u_int32_t            dh_size;
    u_int32_t            dh_defines;  /* Magic number defined */
    struct tr_time_t     dh_ts;
    u_int32_t            dh_ip;       /* host generating stream */
    u_int32_t            dh_dev_type; /* device described */
    struct tr_prop_lst_t dh_plist[0]; /* Variable size */
};

struct tr_dev_ent_t {
    u_int32_t        de_magic;
    u_int32_t        de_size;
    struct tr_time_t de_ts;
    u_int32_t        de_vlist[0]; /* Variable size */
};
```

Figure 5: Device header and entry records

These optional characteristics, listed in Table 2, are mostly concerned with the signal parameters of the wireless interfaces we have available. Interpreting these parameters is heavily device-dependent. We give examples of how we've used device observations in Section 5.

DEV_ID	Major and minor number of device (header only)
DEV_STATUS	Device specific status registers
WVLN_SIGTONOISE	Signal to noise ratio reported by WaveLAN
WVLN_SIGQUALITY	Signal quality reported by WaveLAN
WVLN_SILENCELVL	WaveLAN silence level

Table 2: Current optional fields for packet entries

4.5. Miscellaneous Tracks

We use miscellaneous, or general, tracks to record things that don't fit clearly in either the packet or device model. At the moment, physical location of a mobile host is the only attribute tracked in general trace records. The required portion of the general header and entry records is shown in Figure 6, the two optional properties are in Table 3. In addition to the property list, general headers have only the IP address of the host generating the record and the time at which observations began. General entries have only a timestamp, and the optional fields.

4.6. Annotations

An experimenter may occasionally want to embed arbitrary descriptive text into a trace. We include annotation records to provide for this. Such records are not part of a track; they stand alone. The structure of an annotation record is shown in Figure 7. Annotations include the time at which the annotation was inserted in the trace, the host which inserted the annotation, and the variable-sized text of the annotation itself.

```
struct tr_gen_hdr_t {
    u_int32_t            gh_magic;
    u_int32_t            gh_size;
    u_int32_t            gh_defines;
    struct tr_time_t     gh_ts;
    u_int32_t            gh_ip;
    struct tr_prop_lst_t gh_plist[0]; /* Variable size */
};

struct tr_gen_ent_t {
    u_int32_t        ge_magic;
    u_int32_t        ge_size;
    struct tr_time_t ge_ts;
    u_int32_t        ge_vlist[0]; /* Variable size */
};
```

Figure 6: General header and entry records

MH_LOC_X	Mobile host's X coordinate (map-relative)
MH_LOC_Y	Mobile host's Y coordinate (map-relative)
MH_LOC_LAT	Mobile host's GPS latitude
MH_LOC_LON	Mobile host's GPS longitude

Table 3: Current optional fields for general entries

```
struct tr_annote_t {
    u_int32_t        a_magic;
    u_int32_t        a_size;
    struct tr_time_t a_ts;
    u_int32_t        a_ip;
    char             a_text[0]; /* variable size */
};
```

Figure 7: Annotation records

4.7. Lost Trace Data

It is possible that, during collection, some trace records may be lost due to trace buffer overflow or other reasons. Rather than throw such traces away, or worse, ignoring the lost data, we've included a loss record to count the types of other records which are lost in the course of trace collection. Loss records are shown in Figure 8.

```
struct tr_loss_t {
    u_int32_t        l_magic;
    u_int32_t        l_size;
    struct tr_time_t l_ts;
    u_int32_t        l_ip;
    u_int32_t        l_pkthdr;
    u_int32_t        l_pktent;
    u_int32_t        l_devhdr;
    u_int32_t        l_devent;
    u_int32_t        l_annote;
};
```

Figure 8: Loss records

5. Software Components

In this section, we describe the set of tools that have been built to date for mobile network tracing. We believe many of these tools are widely applicable to network tracing tasks, but some have particular application to mobile network tracing. We begin with an overview of the tools, their applicability, and the platforms on which they are currently supported, as well as those they are being ported to. This information is summarized in Table 4.

We have made every effort to minimize dependencies of our software on anything other than protocol and device specifications. As a result, we expect ports to other BSD-derived systems to be straightforward; ports to other UNIX systems may be more complicated, but feasible.

There are three categories into which our tracing tools can be placed: trace collection, trace modulation, and trace analysis. Trace collection tools are used for generating new traces. They record information about the general networking facilities, as well as data specific to mobile situations: mobile host location, base station location, and wireless device characteristics. These tools are currently supported on BSDI, and are being ported to NetBSD. We describe these tools in Section 5.1.

Trace modulation tools emulate the performance of a traced wireless network on a private wired network. The trace modulation tools, discussed in Section 5.2, are currently supported on NetBSD platforms. They are geared toward replaying low speed/quality networks on faster and more reliable ones, and are thus most applicable to reproducing mobile environments.

In Section 5.3, we conclude with a set of trace processing and analysis tools, which are currently supported on both NetBSD and BSDI platforms. Our analyses to date have focused on properties of wireless networks, and are most directly applicable to mobile traces. The processing tools, however, are of general utility.

	Collection	Modulation	Analysis
NetBSD	In Progress	Supported	Supported
BSDI	Supported	Planned	Supported

This table summarizes the currently supported platforms for the tracing tool suites, and the platforms to which ports are underway.

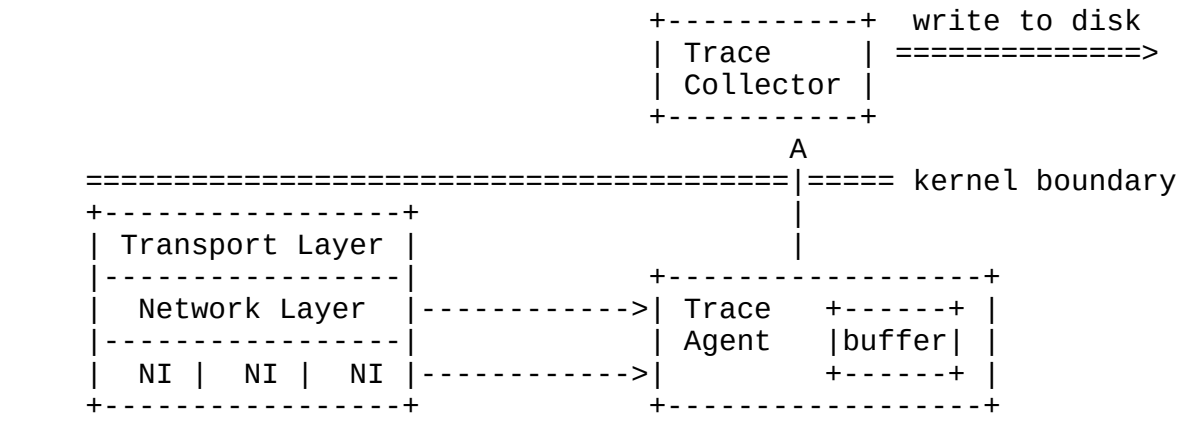
Table 4: Tool Availability

5.1. Trace Collection Tools

The network trace collection facility comprises two key components: the trace agent and the trace collector. They are shown in Figure 9.

The trace agent resides in the kernel where it can obtain data that is either expensive to obtain or inaccessible from the user level. The agent collects and buffers data in kernel memory; the user-level trace collector periodically extracts data from this kernel buffer and writes it to disk. The buffer amortizes the fixed costs of data transfer across a large number of records, minimizing the impact of data transfer on system performance. The trace collector retrieves data through a pseudo-device, ensuring that only a single -- and therefore complete -- trace file is being generated from a single experiment. To provide simplicity and efficiency, the collector does not interpret extracted data; it is instead processed off-line by the post-processing and analysis tools described in Sections 5.2 and 5.3.

There are three sorts of data collected by the tracing tools: network traffic, network device characteristics, and mobile host location. The first two are collected in much the same way; we describe the methodology in Section 5.1.1. The last is collected in two novel ways. These collection methods are addressed in Section 5.1.2.



This figure illustrates the components of trace collection. The NI's are network interfaces.

Figure 9: Components of trace collection

5.1.1. Traffic and Device Collection

The trace agent exports a set of function calls for traffic and device data collection. Traffic data is collected on a per-packet basis. This is done via a function called from device drivers with the packet and a device identifier as arguments. For each packet, the trace record contains the source and destination address options. Since our trace format assembles related packets into tracks, common information, such as the destination address, is recorded in the track header to reduce the record size for each packet entry. We also record the size of each packet.

Information beyond packet size and address information is typically protocol-dependent. For transport protocols such as UDP and TCP, for example, we record the source and destination port numbers; TCP packet records also contain the sequence number. For ICMP packets, we record their type, code and additional type-dependent data. As explained in Section 5.2.3, we record the identifier, sequence number and time stamp for ICMP ECHOREPLY packets.

Before appending the record to the trace buffer, we check to see if it is the first record in a track. If so, we create a new packet track header, and write it to the buffer prior the packet entry.

Our trace collection facility provides similar mechanisms to record device-specific data such as signal quality, signal level, and noise level. Hooks to these facilities can be easily added to the device drivers to invoke these tracing mechanisms. The extensible and self-describing features of our trace format allow us to capture a wide variety of data specific to particular network interfaces.

For wireless network devices, we record several signal quality measurements that the interfaces provide. Although some interfaces, such as NCR's WaveLAN, can supply this of information for every packet received, most devices average their measurements over a longer period of time. As a result, we only trace these measurements periodically. It is up to the device drivers to determine the frequency at which data is reported to the trace agent.

When devices support it, we also trace status and error events. The types of errors, such as CRC or buffer overflow, allow us to determine causes for some observed packet losses. For example, we can attribute loss to either the wireless channel or the network interface.

5.1.2. Location Tracing

At first thought, recording the position of a mobile host seems straightforward. It can be approximated by recording the base station (BS) with which the mobile host is communicating. However, due to the large coverage area provided by most radio interfaces, this information provides a loose approximation at best. In commercial deployments, we may not be able to reliably record the base station with which a mobile host communicates. This section outlines our collection strategy for location information in both outdoor and indoor environments.

The solution that we have considered for wide-area, outdoor environments makes use of the Global Positioning System (GPS). The longitude and latitude information provided by the GPS device is recorded in a general track.

Indoor environments require a different approach because the satellite signals cannot reach a GPS device inside a building. We considered deploying an infrared network similar to the Active Badge [14] or the ParcTab [12]; however, this significant addition to the wireless infrastructure is not an option for most research groups.

As an alternative, we have developed a graphical tool that displays the image of a building map and expects the user to "click" their location as they move; the coordinates on the map are recorded in one or more general tracks. The header of such tracks can also record the coordinates of the base stations if they are known.

An extension can be easily added to this tool to permit multiple maps. As the user requests that a new map be loaded into the graphical tracing tool, a new location track is created along with an annotation record that captures the file name of that image. Locations of new base stations can be recorded in this new track

header. Each location track should represent a different physical and wireless environment.

5.2. Trace Modulation Tools

A key tool we have built around our trace format is PaM, the Packet Modulator. The idea behind PaM is to take traces that were collected by a mobile host and distill them into modulation traces. These modulation traces capture the networking environment seen by the traced host, and are used by a PaM kernel to delay, drop, or corrupt incoming and outgoing packets. With PaM, we've built a testbed that can repeatably, reliably mimic live systems under certain mobile scenarios.

There are three main components to PaM. First, we've built a kernel capable of delaying, dropping, and corrupting packets to match the characteristics of some observed network. Second, we've defined a modulation trace format to describe how such a kernel should modulate packets. Third, we've built a tool to generate modulation traces from certain classes of raw traces collected by mobile hosts.

5.2.1. Packet Modulation

The PaM modulation tool has been placed in the kernel between the IP layer and the underlying interfaces. The tool intercepts incoming and outgoing packets, and may choose to drop it, corrupt it, or delay it. Dropping an incoming or outgoing packet is easy, simply don't forward it along. Similarly, we can corrupt a packet by flipping some bits in the packet before forwarding it.

Correctly delaying a packet is slightly more complicated. We model the delay a packet experiences as the time it takes the sender to put the packet onto the network interface plus the time it takes for the last byte to propagate to the receiver. The former, the transmission time, is the size of the packet divided by the available bandwidth; the latter is latency.

Our approach at delay modulation is simple -- we assume that the actual network over which packets travel is much faster and of better quality than the one we are trying to emulate, and can thus ignore it. We delay the packet according to our latency and bandwidth targets, and then decide whether to drop or corrupt it. We take care to ensure that packet modulation does not unduly penalize other system activity, using the internal system clock to schedule packets. Since this clock is at a large granularity compared to delay resolution, we try to keep the average error in scheduling to a minimum, rather than scheduling each packet at exactly the right time.

5.2.2. Modulation Traces

To tell the PaM kernel how the modulation parameters change over time, we provide it with a series of modulation-trace entries. Each of these entries sets loss and corruption percentages, as well as network latency and inter-byte time, which is 1/bandwidth. These entries are stored in a trace file, the format of which is much simpler than record-format traces, and is designed for efficiency in playback. The format of modulation traces is shown in Figure 10.

```
struct tr_rep_hdr_t {
    u_int32_t        rh_magic;
    u_int32_t        rh_size;
    u_int32_t        rh_time_fmt;         /* nsec or used */
    struct tr_time_t rh_ts;
    char             rh_date[TR_DATESZ];
    char             rh_agent[TR_NAMESZ];
    u_int32_t        rh_ip;
    u_int32_t        rh_ibt_ticks;        /* units/sec, ibt */
    u_int32_t        rh_lat_ticks;        /* units/sec, lat */
    u_int32_t        rh_loss_max;         /* max loss rate */
    u_int32_t        rh_crpt_max;         /* max corrupt rate */
    char             rh_desc[0];          /* variable size */
};

struct tr_rep_ent_t {
    u_int32_t         re_magic;
    struct tr_time_t  re_dur;          /* duration of entry */
    u_int32_t         re_lat;          /* latency */
    u_int32_t         re_ibt;          /* inter-byte time */
    u_int32_t         re_loss;         /* loss rate */
    u_int32_t         re_crpt;         /* corrupt rate */
};
```

Figure 10: Modulation trace format

Modulation traces begin with a header that is much like that found in record-format trace headers. Modulation headers additionally carry the units in which latency and inter-byte time are expressed, and the maximum values for loss and corruption rates. Individual entries contain the length of time for which the entry applies as well as the latency, inter-byte time, loss rate, and corruption rate.

5.2.3. Trace Transformation

How can we generate these descriptive modulation traces from the recorded observational traces described in Section 4? To ensure a high-quality modulation trace, we limit ourselves to a very narrow set of source traces. As our experience with modulation traces is limited, we use a simple but tunable algorithm to generate them.

Our basic strategy for determining latency and bandwidth is tied closely to our model of packet delays: delay is equal to transmission time plus latency. We further assume that packets which traversed the network near one another in time experienced the same latency and bandwidth during transit. Given this, we look for two packets of different size that were sent close to one another along the same path; from the transit times and sizes of these packets, we can determine the near-instantaneous bandwidth and latency of the end-to-end path covered by those packets. If traced packet traffic contains sequence numbers, loss rates are fairly easy to calculate. Likewise, if the protocol is capable of marking corrupt packets, corruption information can be stored and then extracted from recorded traces.

Using timestamped packet observations to derive network latency and bandwidth requires very accurate timing. Unfortunately, the laptops we have on hand have clocks that drift non-negligibly. We have chosen not to use protocols such as NTP [9] for two reasons. First, they produce network traffic above and beyond that in the known traced workload. Second, and perhaps more importantly, they can cause the clock to speed up or slow down during adjustment. Such clock movements can play havoc with careful measurement.

As a result, we can only depend on the timestamps of a single machine to determine packet transit times. So, we use the ICMP ECHO service to provide workloads on traced machines; the ECHO request is timestamped on it’s way out, and the corresponding ECHOREPLY is traced. We have modified the ping program to alternate between small and large packets. Traces that capture such altered ping traffic can then be subject to our transformation tool.

The tool itself uses a simple sliding window scheme to generate modulation entries. For each window position in the recorded trace, we determine the loss rate, and the average latency and bandwidth experienced by pairs of ICMP ECHO packets. The size and granularity of the sliding window are parameters of the transformation; as we gain experience both in analysis and modulation of wireless traces, we expect to be able to recommend good window sizes.

Unfortunately, our wireless devices do not report corrupt packets; they are dropped by the hardware without operating system notification. However, our modulation system will also coerce any such corruptions to an increased loss rate, duplicating the behavior in the original network.

5.3. Trace Analysis Tools

A trace is only as useful as its processing tools. The requirements for such tools tools include robustness, flexibility, and portability. Having an extensible trace format places additional emphasis on the ability to work with future versions. To this end, we provide a general processing library as a framework for users to easily develop customized processing tools; this library is designed to provide both high portability and good performance.

In this section, we first present the trace library. We then describe a set of tools for simple post-processing and preparing the trace for further analyses. We conclude with a brief description of our analysis tools that are applied to this minimally processed data.

5.3.1. Trace Library

The trace library provides an interface that applications can use to simplify interaction with network traces, including functions to read, write, and print trace records. The trace reading and writing functions manage byte swapping as well as optional integrity checking of the trace as it is read or written. The library employs a buffering strategy that is optimized to trace I/O. Trace printing facilities are provided for both debugging and parsing purposes.

5.3.2. Processing Tools

The processing tools are generally the simplest set of tools we have built around the trace format. By far the most complicated one is the modulation-trace transformation tool described in Section 5.2.3; the remainder are quite simple in comparison. The first such tool is a parser that prints the content of an entire trace. With the trace library, it is less than a single page of C code. For each record, it prints the known data fields along with their textual names, followed by all the optional properties and values.

Since many analysis tasks tend to work with records of the same type, an enhanced version of the parser can split the trace data by tracks into many files, one per track. Each line of the output text files contains a time stamp followed by the integer values of all the optional data in a track entry; in this form traces are amenable to further analysis be scripts written in an interpreted language such

as perl.

We have developed a small suite of tools providing simple functions such as listing all the track headers and changing the trace description as they have been needed. With the trace library, each such tool is trivial to construct.

5.3.3. Analysis Tools

Analysis tools depend greatly on the kind of information an experimenter wants to extract from the trace; our tools show our own biases in experimentation. Most analyses derive common statistical descriptions of traces, or establish some correlation between the trace data sets.

As early users of the trace format and collection tools, we have developed a few analysis tools to study the behavior of the wireless networks at our disposal. We have been particularly interested in loss characteristics of wireless channels and their relation to signal quality and the position of the mobile host. In this section, we briefly present some of these tools to hint at the kind of experimentation possible with our trace format.

Loss characteristics are among the most interesting aspects of wireless networks, and certainly among the least well understood. To shed light on this area, we have created tools to extract the loss information from collected traces; in addition to calculating the standard parameters such as the packet loss rate, the tool also derives transitional probabilities for a two-state error model.

This has proven to be a simple yet powerful model for capturing the burstiness observed in wireless loss rates due to fading signals. To help visualize the channel behavior in the presence of mobility, our tool can replay the movement of the mobile host while plotting the loss rate as it changes with time. It also allows us to zoom in the locations along the path and obtain detailed statistics over arbitrary time intervals.

Our traces can be further analyzed to understand the relationship between channel behavior and the signal quality. For wireless devices like the NCR WaveLAN, we can easily obtain measurements of signal quality, signal strength, and noise level. We have developed a simple statistical tool to test the correlation between measured signal and the loss characteristics. Variations of this test are also possible using different combinations of the three signal measurements and the movement of the host.

The question of just how mobile such mobile hosts are can also be investigated through our traces. Position data are provided by traces that either involved GPS or user-supplied positions with our trace collection tools. This data is valuable for comparing and validating various mobility prediction algorithms. Given adequate network infrastructure and good signal measurements, we can determine the mobile location within a region that is significantly smaller than the cell size. We are developing a tool to combine position information and signal measurement from many traces to identify the "signal quality" signature for different regions inside a building. Once this signature database is completed and validated, it can be used to generate position information for other traces that contain only the signal quality information.

6. Related Work

The previous work most relevant to mobile network tracing falls into two camps. The first, chiefly exemplified by tcpdump [7] and the BSD Packet Filter, or BPF [8], collect network traffic data. The second, notably Delayline [6], and the later Probe/Fault Injection Tool [4], and the University of Lancaster’s netowrk emulator [3], provide network modulation similar to PaM.

There are many systems that record network packet traffic; the de facto standard is tcpdump, which works in concert with a packet filter such as BPF. The packet filter is given a small piece of code that describes packets of interest, and the first several bytes of each packet found to be interesting is copied to a buffer for tcpdump to consume. This architecture is efficient, flexible, and has rightly found great favor with the networking community.

However, tcpdump cpatures only traffic data. It records neither information concerning mobile networking devices nor mobile host location. Rather than adding seperate software components to a host running tcpdump to capture this additional data, we have chosen to follow an integrative approach to ease trace file administration. We have kept the lessons of tcpdump and BPF to heart; namely copying only the information necessary, and transferring data up to user level in batches. It may well pay to investigate either incorporating device and location information directly into BPF, or taking the flexible filtering mechanism of BPF and including it in our trace collection software. For the moment, we do not know exactly what data we will need to explore the properties of mobile networks, and therefore do not exclude any data.

There are three notable systems that provide packet modulation similar to PaM. The earliest such work is Delayline, a system designed to emulate wide-area networks atop local-area ones; a goal

similar to PaM's. The most striking difference between Delayline and PaM is that Delayline's emulation takes place entirely at the user-level, and requires applications to be recompiled against a library emulating the BSD socket system and library calls. While this is a portable approach that works well in the absence of kernel-level source access, it has the disadvantage that not all network traffic passes through the emulation layer; such traffic may have a profound impact on the performance of the final system. Delayline also differs from PaM in that the emulated network uses a single set of parameters for each emulated connection; performance remains fairly constant, and cannot change much over time.

The Lancaster network emulator was designed explicitly to model mobile networks. Rather than providing per-host modulation, it uses a single, central server through which all network traffic from instrumented applications passes. While this system also does not capture all traffic into and out of a particular host, it does allow modulation based on multiple hosts sharing a single emulated medium. There is a mechanism to change the parameters of emulation between hosts, though it is fairly cumbersome. The system uses a configuration file that can be changed and re-read while the system is running.

The system closest in spirit to PaM is the Probe/Fault Injection Tool. This system's design philosophy allows an arbitrary protocol layer -- including device drivers -- to be encapsulated by a layer below to modulate existing traffic, and a layer above to generate test traffic. The parameters of modulation are provided by a script in an interpreted language, presently Tcl, providing considerable flexibility. However, there is no mechanism to synthesize such scripts -- they must be explicitly designed. Furthermore, the use of an interpreted language such as Tcl limits the use of PFI to user-level implementations of network drivers, and may have performance implications.

7. Future Work

This work is very much in its infancy; we have only begun to explore the possible uses for mobile network traces. We have uncovered several areas of further work.

The trace format as it stands is very IP-centric. While one could imagine using unknown IP addresses for non-IP hosts, while using header-only properties to encode other addressing schemes, this is cumbersome at best. We are looking into ways to more conveniently encode other addressing schemes, but are content to focus on IP networks for the moment.

Two obvious questions concerning wireless media are the following. How does a group of machines perform when sharing the same bandwidth? How asymmetric is the performance of real-world wireless channels? While we do have tools for merging traces taken from multiple hosts into a single trace file, we've not yet begun to examine such multiple-host scenarios in depth. We are also looking into instrumenting wireless base stations as well as end-point hosts.

Much of our planned work involves the PaM testbed. First and foremost, many wireless channels are known to be asymmetric; splitting the replay trace into incoming and outgoing modulation entries is of paramount importance. We would like to extend PaM to handle multiple emulated interfaces as well as applying different modulation parameters to packets from or to different destinations. One could also imagine tracing performance from several different networking environments, and switching between such environments under application control. For example, consider a set of traces showing radio performance at various altitudes; an airplane simulator in a dive would switch from high-altitude modulation traces to low-altitude ones.

Finally, we are anxious to begin exploring the properties of real-world mobile networks, and subjecting our own mobile system designs to PaM to see how they perform. We hope others can make use of our tools to do the same.

Acknowledgements

The authors wish to thank Dave Johnson, who provided early pointers to related work and helped us immeasurably in RFC formatting. We also wish to thank those who offered comments on early drafts of the document: Mike Davis, Barbara Denny, Mark Lewis, and Hui Zhang. Finally, we would like to thank Bruce Maggs and Chris Hobbs, our first customers!

This research was supported by the Air Force Materiel Command (AFMC) and ARPA under contract numbers F196828-93-C-0193 and 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, ARPA, AT&T, Hughes, IBM, Intel, Metricom, Carnegie Mellon University, the University of California, the State of California, or the U.S. Government.

Security Considerations

This RFC raises no security considerations.

Authors' Addresses

Questions about this document can be directed to the authors:

Brian D. Noble
Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

Phone: +1-412-268-7399
Fax: +1-412-268-5576
EMail: bnoble@cs.cmu.edu

Giao T. Nguyen
Room 473 Soda Hall #1776 (Research Office)
University of California, Berkeley
Berkeley, CA 94720-1776

Phone: +1-510-642-8919
Fax: +1-510-642-5775
EMail: gnguyen@cs.berkeley.edu

Mahadev Satyanarayanan
Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891

Phone: +1-412-268-3743
Fax: +1-412-268-5576
EMail: satya@cs.cmu.edu

Randy H. Katz
Room 231 Soda Hall #1770 (Administrative Office)
University of California, Berkeley
Berkeley, CA 94720-1770

Phone: +1-510-642-0253
Fax: +1-510-642-2845
EMail: randy@cs.berkeley.edu

References

[1] Chen, J. B., and Bershad, B. N. The Impact of Operating System Structure on Memory System Performance. In Proceedings of the 14th ACM Symposium on Operating System Principles (Asheville, NC, December 1993).

[2] Dahlin, M., Mather, C., Wang, R., Anderson, T., and Patterson, D. A Quantitative Analysis of Cache Policies for Scalable Network File Systems. In Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (Nashville, TN, May 1994).

[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] Gloy, N., Young, C., Chen, J. B., and Smith, M. D. An Analysis of Dynamic Branch Prediction Schemes on System Workloads. In The Proceedings of the 23rd Annual International Symposium on Computer Architecture (May 1996).

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

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

[8] 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).

[9] Mills, D. L. Improved Algorithms for Synchronizing Computer Network Clocks. IEEE/ACM Transactions on Networking 3, 3 (June 1995).

[10] Mummert, L. B., Ebling, M. R., and Satyanarayanan, M. Exploiting Weak Connectivity for Mobile File Access. In Proceedings of the 15th Symposium on Operating System Prinicples (Copper Mountain, CO, December 1995).

[11] Nelson, M. N., Welch, B. B., and Ousterhout, J. K. Caching in the Sprite Network File System. ACM Transactions on Computer Systems 6, 1 (February 1988).

[12] Schilit, B., Adams, N., Gold, R., Tso, M., and Want, R. The PARCTAB Mobile Computing System. In Proceedings of the 4th IEEE Workshop on Workstation Operating Systems (Napa, CA, October 1993), pp. 34--39.

[13] Uhlig, R., Nagle, D., Stanley, T., Mudge, T., Sechrest, S., and Brown, R. Design Tradeoffs for Software-Managed TLBs. ACM Transactions on Computer Systems 12, 3 (August 1994).

[14] Want, R., Hopper, A., Falcao, V., and Gibbons, J. The Active Badge Location System. ACM Transactions on Information Systems 10, 1 (January 1992), 91--102.