A TRACE-BASED APPROACH TO EVALUATING WIRELESS NETWORKS

Giao T. Nguyen

Computer Science Division

University of California at Berkeley

Berkeley, CA 94720-1776

Table of Content

Abstract

Evaluating a mobile wireless network is a challenging problem because the quality of the wireless link varies unpredictably over time and space. In designing a better mobile computing system, we need proper understanding of the behavior of the wireless links, mobility patterns, and the protocols at various levels of the network hierarchy. This thesis describes a trace-based approach to study such complex systems. Our approach combines the trace collection, modeling, and model validation. We have implemented the trace collection tools for many devices in our testbed. Focusing on the WaveLAN interface, we collect and analyze the signal strength, the packet error characteristics, and their correlation. We also derive an empirical model and derive parameters for the two-state Markov model. For validation, we implement these models and a trace-replay mechanism in the network simulator *ns*. We then compare the TCP performance of the network subjected to errors generated from the models and the traces.

1 Introduction

With the proliferation of portable computers and wireless technologies in recent years, many researchers have found that some conventional protocols do not perform well on the wireless network due to the high loss characteristic [CI95]. This has led to a growing interest in characterizing the channel behavior of many wireless technologies, including wireless LANs and packet radio networks. While it is well known that wireless links have higher error rates than their wired counterparts, the wireless error characteristics are not well understood due to their inherent dependence on the complex radio wave propagation. Characterizing the loss behavior is an important problem since it is one of the few key parameters that affect all levels of the network stack.

Due to the large number of protocols at different network layers, it is often impractical to build and evaluate all of them. A better alternative is to trace the behaviors of existing implementations and apply the lessons learned to evaluate new designs. From wireless network traces [NNSK96], we can improve our understanding of the wireless channel behavior, develop realistic models, and validate them. Traces also provide us a way to reproduce realistic network and mobile environments for comparing protocols and algorithms [NSNK96].

The focus of this thesis is on the tracing and modeling of wireless channel errors. Although the uniform bit error rate model has often been used to simulate lossy links, it does not accurately represent the wireless channels. Our goal is to find an error model that is realistic yet simple to implement. The models that we investigate include the two-state Markov model [KS76], and the empirical model. From the empirical distributions of the error and error-free length, we also derive the mathematical expressions that fit our data more closely than the two-state Markov model.

Although tracing is the most important part of our work, we also need simulation for validating the error models. Our simulator is an extension the network simulator *ns* [MF95], originally developed by LBL. *Ns* is a discrete event simulator with the core components implemented in C++ and the configuration interface written in Tcl. To simulate wireless and local area networks (LAN) in *ns*, we added several new modules that implement the medium access control (MAC), the physical channel, the trace-replay and loss models.

The remainder of this paper is organized as follows. In Section 2, we provide some background on wireless errors and the testbed for this study. Our trace-based approach consists of three phases, described in three separate sections along with the associated data. Section 3 introduces our methodology and describes the trace collection process. Section 4 presents the analysis of our packet error and signal strength data. In Section 5, we analyze and model the distributions of packet errors. Section 6 describes the simulation environment that we used to validate our models. In Section 7, we discuss validity of our models based on our simulation results.

2 Background

Wireless communication errors are mainly caused by the inability of the receiver to distinguish the transmitted signal from the background noise. At the physical level, radio waves propagate using three basic mechanisms: reflection, diffraction, and scattering. The resulting effects of these different propagation patterns are free-space loss, shadowing, and multipath fading. Their effect on receiver the signal-to-noise ratio (SNR) has been studied for both the in-building and outdoor environments [ARY95], [CL75], [ES96]. The sensitivity of TCP to wireless errors has prompted many efforts to improve its performance [BB95], [BSK95]. Since wireless errors depend on the environment and the network devices, characterizing them requires some knowledge of the testbed. The following subsections describe our testbed and the wireless interface that we use in this study.

2.1 Testbed

The measurements presented in this paper were collected from the BARWAN wireless overlay network, which is built from many commercially available networking technologies. The available network interfaces include 915 MHz and 2.4 GHz AT&T WaveLAN, Metricom, CDPD, infrared, and DirecPC. The mobile hosts (MH) are IBM Thinkpad laptops and the base stations (BS) are Dell PCs. Both the mobile hosts and base stations are x86 computers running a common UNIX operating system, BSD/OS Version 2.1. Although the base stations and mobile hosts can have multiple wireless interfaces, we only evaluate the 915

MHz WaveLAN in this study. Having complete control over this WaveLAN network allows us to obtain many device-specific data that are valuable for channel modeling.

2.2 WaveLAN

WaveLAN is a commercial wireless network interface operating in the 902-928 MHz ISM (Industrial, Scientific, and Medical) band [AT&T93], [KMM95]. This Direct Sequence Spread Spectrum (DS-SS) modem has a spreading factor of 11 chips per bit and a maximum bandwidth of about 2 Mbps. WaveLAN does not implement any error recovery mechanism, making it appropriate for studying wireless errors.

Medium access control (MAC) in WaveLAN resembles that of an Ethernet. Being a CSMA scheme, it constantly monitors the state of the wireless channel. Before each transmission, the MAC must wait until the end of the current transmission plus a small delay known as the WaveLAN Inter-Frame Space (WIFS). Since it is difficult to detect a collision while transmitting, WaveLAN implements a collision avoidance scheme, which is different from the CSMA/CA being proposed by the IEEE 802.11 committee [C94]. After the WIFS, the MAC waits for a random number of *antenna slots* less than an initial back-off value. If the channel becomes busy before its slot, the MAC doubles its back-off window and continues to monitor the channel for the next contention period. This back-off process is repeated for a maximum of 15 times before the packet is dropped.

The WaveLAN link is slightly asymmetric due to differences in implementation of the base station's ISA interface and the mobile host's PCMCIA counterpart. The ISA adapter has an Intel 82586 CSMA/CD LAN controller, while the PCMCIA version has the newer 82593 chip. Although both interfaces have 0.5 nanosecond bit-times and 32-bit inter-frame space, the length of the antenna slot for back-off purpose is significantly different: 12 bit-times for ISA and 32 bit-times for PCMCIA. To partially make up the longer antenna slot, the PCMCIA controller sets its initial back-off window to 16 slots (512 bit-times) instead of the ISA's 32 (384 bit-times). The larger average delay for PCMCIA leads to lower throughput.

Like many other wireless interfaces, WaveLAN provides access to the signal strength information. The three signal measurements for each packet are signal level, noise level, and signal quality. The signal level is a 6-bit measurement derived from the receiver's automatic gain control (AGC) setting at the beginning of each transmission. Similarly, the noise level is measured immediately after the end of the transmission, when no WaveLAN interface within detectable range can transmit. The signal quality is the 4-bit measurement derived from the information used for antenna selection at the beginning of packet reception.

3 Methodology

Modeling the wireless channel is challenging because its quality can vary dramatically over time and space. To deal with this problem, we use a trace-based approach consisting of three phases: trace collection, modeling, and validation. In the trace collection phase, a large number of traces are collected for many different scenarios. The modeling phase involves extracting the data of interest, such as packet errors, and modeling them. In the validation phase, the models are simulated and compared with the trace data.

This section concentrates on the trace collection phase, leaving the modeling and validation phases for later sections. To facilitate trace collection, we have developed many tools including a general purpose trace engine, a graphical user interface, and a workload generator. Our trace engine implements an extensible trace format capable of recording information such as location and network traffic. Appendix A provides an overview of our trace format. Along with the core tracing tools, we also provide a graphical user interface that can record and animate the location information along with the corresponding network conditions. In the following subsections, we describe these tracing tools and the workload generation.

3.1 Trace Collection

The network tracing facility for both the base station and the mobile host contains two key components: the *trace agent* and the *trace collector* as shown in Figure 1. The trace agent runs in the kernel where it can record data that is either inaccessible or expensive to obtain at the user level. We added kernel hooks to the device driver of the interested network interface (NI) to pass the packet and device-level information to the trace agent. These data are periodically extracted from the kernel buffer and saved to disk by the trace collector, which runs as a user-level process. Both the trace agent and trace collector are transparent to the network stack and applications.

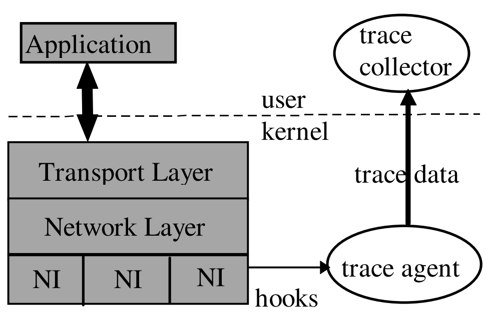

Figure 1. Network Trace Collection

When the trace collection program is executed, it opens a network tracing pseudo-device to start the kernel trace agent. The kernel trace agent then puts the wireless network interfaces into promiscuous mode to capture all packets. This mechanism is similar to the Berkeley Packet Filter [MJ93] widely used by the network community. The main difference is in our trace format, which enables us to record other information besides the packet headers. Device-specific data such as signal strength is valuable in our analysis of the wireless channel.

3.2 Recording and Visualizing Mobility

Although our trace engine can record the location information, obtaining this information is not trivial. The location of the mobile host can be approximated by tracking the closest base station. This provides very coarse approximation because each radio base station covers a large area. We can achieve better accuracy by using an infrared network similar to the Active Badge [WHFG92] or the ParcTab [SA+93] system, but this is a costly addition to the wireless infrastructure. As an alternative, our graphical tool displays a map and records the relative location supplied by the user.

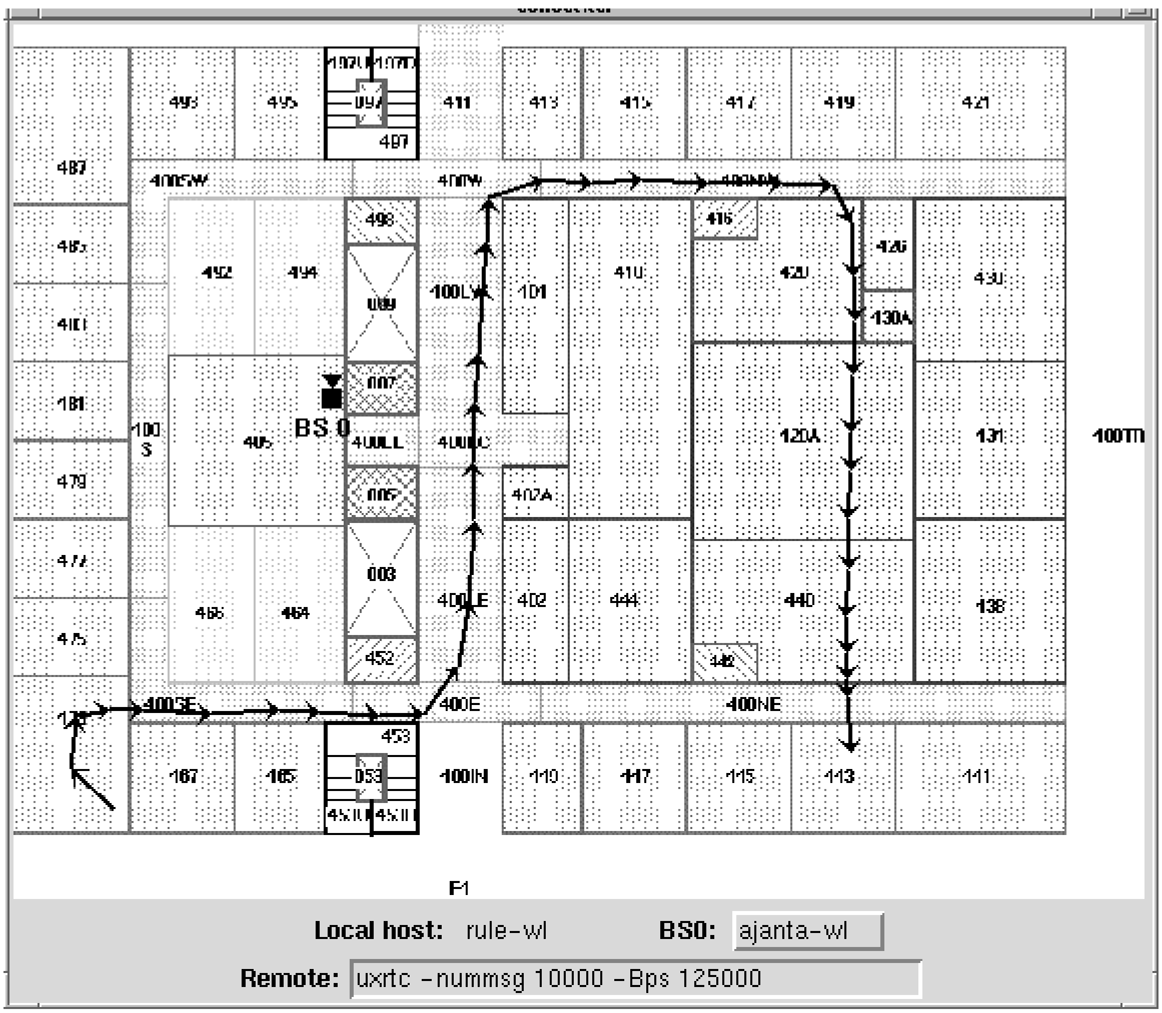

Figure 2. Location Recording Tool: arrows depict the path taken by the user/MH

Figure 2 shows the snap-shot the location recording tool displaying a map of the fourth floor of Soda Hall at UC Berkeley, which is the environment for our trace collections. This tool runs on the mobile host and expects the user to "click" on the map while moving. It records the location of the mobile host along with the network and device information into a single trace file.

Complementing the location recorder is the visualizing tool. It processes the trace and replays the information collected by the location recorder. The visualizing tool has two main windows. The first window displays the map and animates the path of the mobile host. The second window plots the time-varying statistics including the packet error rate, signal level, noise level, and signal quality. The visualizing tool can display the statistics for parts of the trace or over the entire trace. It also offers other capabilities including printing, executing other trace processing and analysis tools. These capabilities help the user to quickly analyze and understand the dynamic behavior of the wireless network.

3.3 Workload Generation

Besides the tracing tools, we need a flexible workload generator to create interesting network conditions for tracing. We implement a UDP traffic generator capable of varying the transmission rate and packet size. For error measurements, UDP is preferred over TCP because it has no error-recovery and connection establishment mechanisms. Each UDP packet is specially formatted to include information for error detection, such as a sequence number. For our modeling of packet errors, we fixed the transmission rate and packet size such that our models can be converted into time-based ones. The effects of these two parameters are discussed in the following subsections.

To simplify the analysis process, each trace captures a single traffic stream. We also try to reduce the interference of the trace workload and other traffic. Although UDP does not require the presence of the listener on the destination port, it will send "port unreachable" error messages back to the sender. These error messages interfere with the traffic being traces. To avoid this problem, we can suppress the error messages in the kernel or have a user-level listener. The latter incurs more overhead for copying all packets from the kernel into the user-level memory.

4 Experimental Results

Using the trace collection tools, we collected a large number of traces at both the BS and MH inside Soda Hall at UC Berkeley. The MH is either stationary or moving but the BS is always at a fixed location. In the following subsections, we analyze the packet errors in term of their correlation with the transmission rate, the packet size, the distance between the BS and MH, and the received signal strength.

4.1 Effect of Transmission Rate

To account for the asymmetric nature of WaveLAN, and most wireless networks in general, we traced each direction of the link separately. Our measurements indicate that the base station's ISA interface can send large UDP packets of more than 1000 bytes at a maximum throughput of about 1.6 Mbps. Using the same packet size, the PCMCIA interface achieves only 1.2 Mbps. This lower transmission rate is mainly due to the differences in implementation of the radio interfaces pointed out in section 2.2.

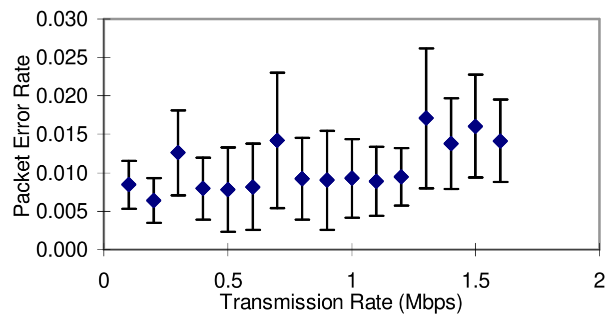

Figure 3. Packet Error Rate vs. Transmission Rate

To investigate the effect of the transmission rate, we used UDP streams at data rate ranging from 0.1 to 1.6 Mbps. We fixed the packet size to 1024 bytes and distance between the transmitter and receiver to 70 feet. Figure 3 plots the average packet error rates for this experiment. For each multiple of 0.1 Mbps in transmission rate, we conducted 10 trials, each involving the transfer of 80 Mb. We compute the packet error rate for each trial, and then find the overall average and standard deviation. The error bars are ± 1 standard deviations from the means. Although we observed slightly higher error rates at higher transmission rates, the large deviations in error rate make this observation inconclusive.

4.2 Effect of Packet Size

After the exploring the transmission rate, we conducted a similar experiment to examine the effect of packet size on error rate. In this experiment, we fixed the transmission rate at 1.5 Mbps and varied the packet size from 100 bytes to 1400 bytes, in 100-byte increments. All packet size was chosen to be less than the maximum transfer unit (MTU) of 1500 bytes for WaveLAN to avoid fragmentation at the IP layer.

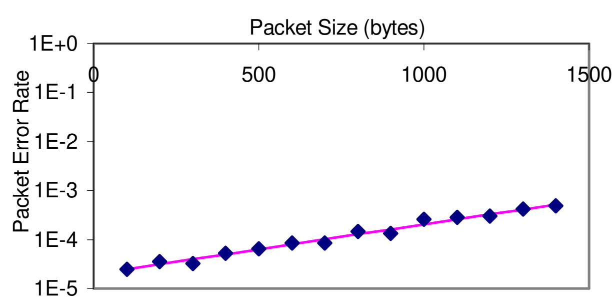

Figure 4. Packet Error Rate vs. Packet Size

The resulting packet error rates are plotted on the semi-log graph in Figure 4. Since the points in the graph lie on a straight line, we infer that the packet error rate increases exponentially with the packet size. Our regression analysis shows that packet error rate doubles for every 300-byte increment of the packet size.

4.3 Effect of Signal Strength

Signal strength is an important parameter of wireless transmissions and has direct impact on the success probability of receiving packets. The reason for this is very simple: signal strength dictates how well the radio interface can detect the transmission. If bits cannot be detected, there will be errors in the packets being received or entire packets might be corrupted. Successful reception of a packet also depends on other factors, such as, the interference with other radio signals and the natural background noise. These sources of interference are often summarized as the noise level at the receiver. If the intended signal is not strong enough to overcome the relative noise level, errors also occur.

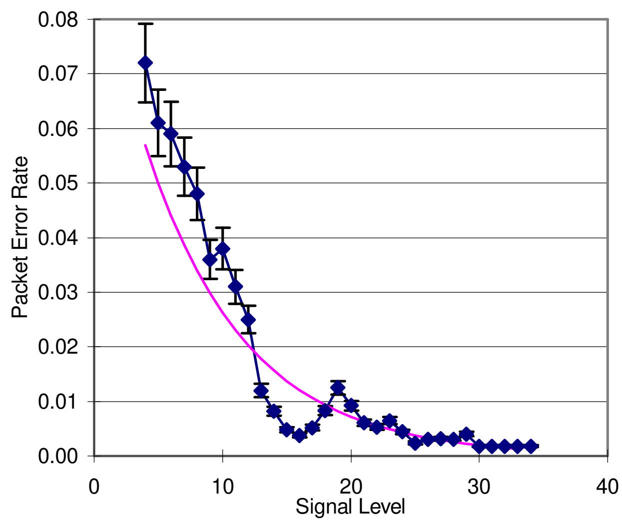

Figure 5. Packet Error Rate vs. Signal Level

To study the effect of signal strength on packet error rate, we collect the signal strength along with the packet-level information. We compute the packet error rate and average signal level over the one-second interval. The average signal level is rounded of to the closest integer before we average the packet error rate corresponding to each distinct signal level.

Figure 5 represents about 43,000 seconds of data collected from many locations on the fourth floor of Soda Hall. The signal level numbers are specific to WaveLAN, which does not document the unit of measurement. The number of samples is not evenly distributed across all signal levels due to the inevitable vagaries of user mobility. As expected, we have higher packet error rates at lower signal levels. We also observe that the error rates increase significantly when the signal level drops below 14. Knowing this dropoff point is relevant to handoff policies that based their actions on the received signal strength. We also fit these data to a simple mathematical expression model, providing a compact form for the observed behaviors. As demonstrated qualitatively in the graph, an exponential decaying curve is a good fit.

4.4 Effect of Distance

Another parameter that characterizes high correlation between signal level and error rate is the distance between the transmitter and receiver. In free space, the power of electromagnetic radiation varies inversely with the square of distance, making distance a strong indicator of signal level as well as loss rate. In practice, the imperfect propagation environment can render the inverse-squared relationship inapplicable.

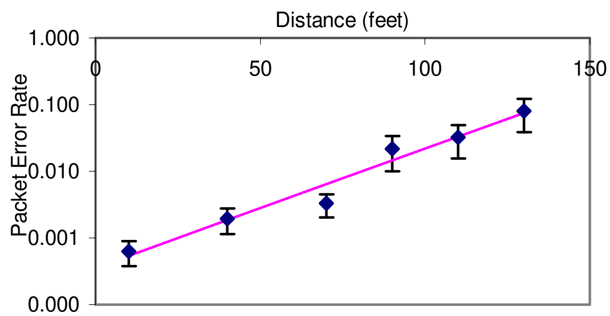

Figure 6. Packet Error Rate vs. Distance

Figure 6 plots the packet error rate measured at various distances in the building. Each data point represents the packet error rate observed in a single room. We conducted three trials, each involving the transfer of 80 Mb. From the plot, we infer that packet error rate is an exponential function of distance. The regression analysis reveals that the packet error rate doubles for every increase of 17 feet within Soda Hall.

	packet error rate		error length		error-free length	
Distance	mean	stdev	mean	stdev	mean	stdev
10	0.0006	0.0002	1.000	0.000	1749.59	732.72
40	0.0020	0.0007	1.848	0.227	1023.43	334.40
70	0.0033	0.0011	2.005	0.210	574.56	148.45
90	0.0218	0.0104	2.428	0.018	122.89	58.99
110	0.0322	0.0148	2.540	0.482	87.26	39.75
130	0.0804	0.0371	2.298	0.290	28.90	8.82
average	0.0247	0.0316	2.054	0.564	549.37	699.39
mobile	0.0327	0.0193	2.365	0.089	84.60	37.36

Table 1. Parameters of the Distance Experiment

Table 1 summarizes the data for our experiment with distance. It contains the mean and standard deviation of the three key packet error characteristics: error rate, error length, and error-free length. The error length is defined to be the number of packets that are lost consecutively. Similarly, the error-free length is number of packets that are successfully received between two adjacent bursts of error. This can also be translated into the inter-arrival time of errors.

The distance of less than 20 feet represents the single room scenario, where the sender and receiver have line-of-sight (LOS). The "average" row contains the averages of the data collected for all the distances. Data in the "mobile" row are collected while the experimenter is moving at the speed of about 5 feet per second inside our building. The packet error rate for this case is about 30% higher than the average, presumably because the presence of mobility produces more errors. The average error length is 2 to 3 packets for most distances. The only noticeable exception is the same-room LOS scenario, which produces only single-packet error bursts. The average error-free lengths for different distances vary by two orders of magnitude and have large standard deviation.

5 Modeling Wireless Errors

After analyzing the packet error rates in the previous section, we turn our attention to the error burstiness. Wireless errors have been empirically observed to be bursty due to various fading effects. As the result, evaluating wireless network protocols with a uniform error model will likely produce inaccurate results. We investigate several models intended to capture the burstiness of wireless errors. From Fritchman's binary error model [F67], we derived two models for our trace data. The first one is the commonly used two-state Markov model. The second one is an empirical model, which is an attempt to fit the distributions more closely. In the following subsection, we describe the two-state Markov model. Following that is the analysis of the error and error-free distributions to improve the two-state model.

5.1 Two-state Markov Model

The basic error model contains two states: error and error-free. When the channel is in the error state, any packets sent would be either lost or corrupted. The opposite is true for the error-free state. In a Markov model, the duration of each state is expressed in term of transition probabilities, as shown in Figure 7. This is the simplest form of the multiple-state Fritchman model.

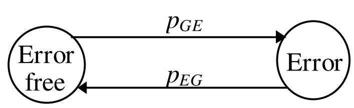

Figure 7. Two-state Error Model

In our derivation of parameters for the two-state Markov model, we label the error state E and error-free state G. The associated transitional probabilities, p_{GE} and p_{EG}, can be computed from the average error-free length L_G and error length L_E. Since both L_E and L_G are geometrically distributed, the transitional probabilities can be expressed by the following formulas [J91]:

$$p_{EG} = \frac{1}{L_E}$$

$$p_{GE} = \frac{1}{L_G}$$

Because the Markov model is memory-less, the transitions can be determined on an individual packet basis. By substituting the *1-F(x)* of the geometric distribution with a random number *u* uniformly distributed from 0 to 1, the length *x* of staying in a state with the leaving probability *p* is:

$$x = \frac{\log(u)}{\log(1-p)}$$

Table 2 provides the error and error-free length averaged from about 30 1000-second traces that are used for modeling purpose. The transmission rate and packet size are fixed at 1.5 Mbps and 1400 bytes respectively, while the distance changes. Since the standard deviations for these parameters are relatively large compared to the means, this model will likely be inaccurate. This motivates us to find more accurate models without sacrificing complexity. The following section describes one such model for the same set of trace data.

	Error	Error-free
probability	0.382	0.006
length (pkts)	2.618	166.394
stdev of length	2.959	496.900
stdev of model	2.058	165.894

Table 2. Parameters for two-state Markov Model

5.2 Empirical Model

Although the two-state Markov model can describe burstiness more accurately than the uniform error model, it cannot capture all loss behavior we have observed in our traces. The main limitation is the underlying assumption that the length distribution for each state is geometric. Our data indicates that neither the error and error-free length distributions is geometric.

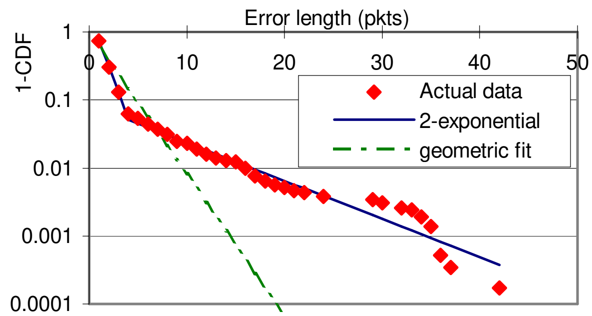

Figure 8. Error Length Distribution

The error length distribution is plotted on a semi-log graph in Figure 8. The dash line is the geometric fit for this data set. Our key observation is that almost 90 percent of all error bursts have length of less than 4 packets. This suggests that we should split error length distribution into two segments. Each segment is fitted with an exponential curve (straight lines on the semi-log graph). Since a distribution must intersect the origin, (0,1), but the best-fit curves do not, each exponential term must be multiplied by a constant factor e^k. The resulting cumulative density function (CDF) and its inverse that expresses the discrete length of error are:

$$1 - F(x) = e^{-x/a} e^{k}$$

$$x = \lceil a(k - \ln(1 - F(x)) \rceil$$

Since these expressions are derived from those of the exponential distribution, we will label them *Exponential**. The parameters a and k for each segment can be separately obtained using linear regression on x and natural log of $1\text{-}F(x)$. For this type of segmented curves, the splitting points must also be specified. These are the 1-CDF values since we are interested it the inverse function. The parameters for the error length distribution are presented in Table 3.

The characteristic error-free length distribution is significantly different from that of the error length. In Figure 9, the 1-CDF is plotted on a log graph to provide more detail of the at the lower range. To avoid having an excessively complicated model, we decided to break this curve into 3 segments: 1 to 37, 37 to about 330, and the rest. The data points for the first two segments seem to fall on two different straight lines. Since the points of these segments fall on straight lines in this graph, they can be expressed by the following formula:

$$1 - F(x) = x^{-a} k$$

This expression describes the Pareto distribution. The inverse function that describes the error-free length is:

$$x = \left[\left(\frac{1}{k}(1 - F(x)) \right)^{-\frac{1}{a}} \right]$$

The parameters a and k for the first two segments can be separately obtained by applying linear regression on the natural log of both x and $(1-F(x))$. The third segment of the error-free length distribution is a concave curve. On a log graph, this type of curve implies exponential decaying function. Therefore, we use the *Exponential** expressions presented earlier to describe this segment. In Figure 9, our best fit curve with 3 segments is close enough to the actual distribution that it is almost invisible.

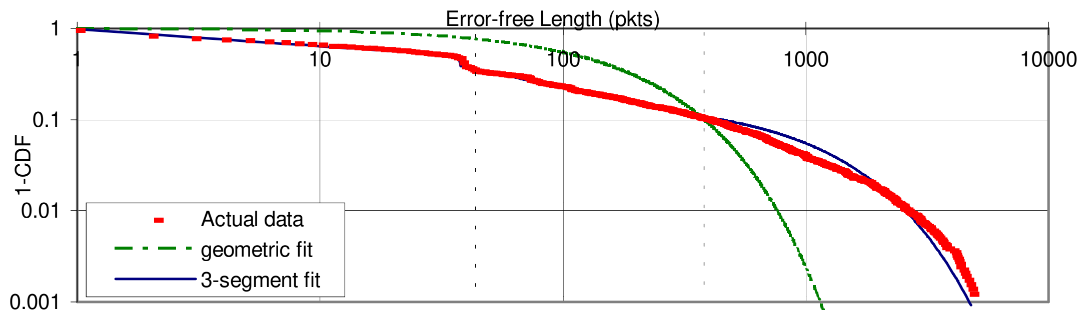

Figure 9. Error-free Length Distribution

Table 3 summarizes all the computed parameters for our empirical model. This model requires 13 parameters: five pairs of a and k, and three 1-CDF values. For each pair a and k obtained from the regression of the cumulative distribution, there is an associated coefficient of determination R^2. This value is the ratio of the regression sum of squares (SSR) to the total sum of squares (SST), which is an indication for the quality of fit. Our model fits the experimental data with coefficients of determination greater than 90%.

BS to MH	segment	Distribution	1-CDF	a	k	R2
Error-free	1 to 37	Pareto	1.000	0.183	0.978	0.974
	38 to 330	Pareto	0.504	0.567	3.053	0.996
	331+	Exponential*	0.113	925.719	-1.820	0.934
Error	1 to 3	Exponential*	1.000	1.153	0.563	1.000
	4+	Exponential*	0.051	7.753	-2.461	0.938

Table 3. Parameters of the Empirical Model for the Downlink

MH to BS	segment	Distribution	1-CDF	a	k	R2
Error-free	1 to 36	Pareto	1.000	0.217	1.033	0.911
	37 to 300	Pareto	0.474	0.781	7.011	0.993
	301+	Exponential*	0.079	441.525	-1.859	0.936
Error	1 to 3	Exponential*	1.000	1.153	0.563	1.000
	4+	Exponential*	0.042	5.145	-2.392	0.929

Table 4. Parameter of the Empirical Model for the Uplink

Wireless loss behavior for the uplink closely resembles that of downlink. The empirical model for the uplink channel also contains two *Exponential** expressions for the error length. For the error-free length, we use a combination of two *Pareto* expressions and one *Exponential** expression. Table 4 provides the model parameters computed for the uplink. Since the error and error-free distributions are very similar for both directions, we have higher confidence that the errors really occur in the wireless channel.

6 Simulation Environment

The accuracy of a simulator depends strongly on its models. Although the quality of the underlying models is critical, several other important factors contribute to a strong simulator design. First, the simulator should be modular, which makes it easier to add new components. Second, it should provide capability to configure simulations at many levels of detail. Finally, the simulator should be efficient. Since many network simulators are available, it is better for us to extend an existing simulator rather than developing one from scratch. Considering these criteria, we choose the network simulator *ns* [MF95] as the basis for our simulation environment.

Ns has an extensible simulation engine implemented in C++ and a Tcl interface for configuration and control. The core of the modular simulator consists of a discrete event scheduler and many components of the network. The *traffic* module provides various types of traffic sources including constant bit rate (CBR), bursty, FTP, and telnet. The transport module implements several TCP flavors and options including Tahoe, Reno, delayed and selective ACKs. The *node* module implements IP unicast and multicast routing. The *link* module contains several queuing disciplines including drop-tail, random early detection (RED) [FJ93], and class-based queuing (CBQ) [FJ95]. The link module also incorporates a simple transmission mechanism to simulate the transmission on wired links.

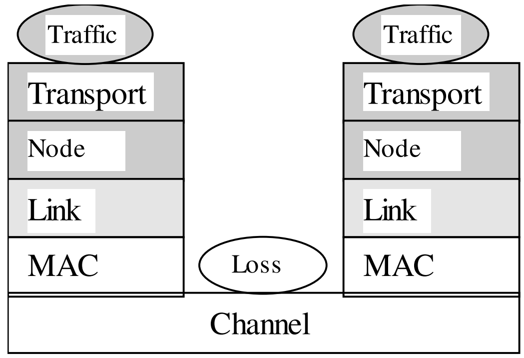

Figure 10. Enhanced structure of *ns*

To simulate the wireless network, we have to extend the structure of the simulator to include the entire network stack similar to those present in real systems. The restructured link module implements only the queuing disciplines. The process of transmission is handled by two new modules: *MAC* and *channel*. The new structure of *ns* is shown in Figure 10. The shaded areas indicate the existing modules that have been modified. The addition of the *MAC*, *channel*, and *loss* modules allows us to simulate the behavior of the channel and its interaction with the MAC protocols. We now describe the functionality of these modules with some implementation insights.

The channel module is the centerpiece of our enhancements to *ns*. It implements a shared medium that carries the transmission from one node to another. Without a shared medium, it is hard to simulate the behavior of a wireless channel. When the channel module receives the packet from the MAC layer, it delivers the packet to the destination node after the channel propagation delay and the transmission time. It also schedules the end-of-transmission event, **`EventSendDone`**, that allows the source host to continue. A collision occurs if the transmission time of one packet overlaps with that of another packet. Once that occurs, the channel marks the packet as corrupted, and issues the collision event, **`EventCollision`**, to all the source hosts that cause the collision. This event is handled by MAC protocols that can detect and deal with collisions. To support CSMA MAC, the channel provides a mechanism for the MAC to check the state of the channel.

The MAC module works in conjunction with the channel module to facilitate the packet transmission. Our MAC module accepts packets from the link module and transmits them on the channel at the appropriate time. We have implemented several MAC protocols such as Aloha, CSMA and CSMA/CD. The Aloha MAC protocol is very simple. It sends out packets when the channel becomes idle. Since the MAC module does not queue packets, it only accepts packets from the link modules after its previous transmission process overhead has transpired. The MAC module also receives packets from the channel and delivers them to the link module after a delay that models receive overhead.

Another widely used MAC protocol that we implement is Carrier Sense Multiple Access (CSMA). Our CSMA MAC object contains several parameters that can be specified to model a variety of CSMA-based interface including WaveLAN and Ethernet. Since CSMA backs off when the channel is busy, it must time out by scheduling itself an event, **`EventSendDone`**. It handles this event by repeating the entire arbitration process including carrier sense and possible back-off. Simulating the Ethernet requires a collision detection mechanism specified by the CSMA/CD protocol. Our implementation of CSMA/CD MAC can handle the collision events raised by the channel.

The loss module is another key addition to *ns* that allows us to effectively simulate the behavior of the wireless channels. This module implements many loss models that can be linked to the channel. The basic loss model has an error rate parameter that can be applied on a bit, byte, or packet. To simulate burst errors, we implement the multiple-state error model. Each state transition in this model can be described by one or more statistical distribution such as exponential, geometric, normal, and Pareto. We also provide another loss object called Empirical, that can read the empirical error distributions from files. This allows us to specify error characteristics that are too complicated to derive mathematical expressions. Finally, we implemented a trace-driven error model. This trace-driven loss object can be configured to extract errors from the traces and replay them in the channel.

7 Validation

Validation is the last of the three phases for our trace-based approach to modeling. Since our primary use of the channel models is to evaluate the impact of the errors on higher-layer network protocols, we choose TCP throughput as the metric for validating our models. We validate the models in the network simulator *ns*. In the following subsections, we describe the simulation of errors in *ns* and provide a quantitative comparison of all the results.

7.1 Simulating Packet Errors

Our simulation setup consists of 2 WaveLAN nodes. A 1000-second file transfer using TCP is done from the sender to the receiver. In this setup, we simulate three error models: uniform packet error rate, two-state Markov, and our empirical model. The uniform packet error rate of 0.01549 is computed from the average error and error-free length. Other parameters for the two-state Markov and the empirical models are taken from Table 2 and Table 3.

Trace modulation contains two steps. The first step is extracting the time and duration of the errors from the traces. In the second step, the errors are replayed in the channel at the simulated time corresponding to the traces. For the validation purpose, we collect 10 new traces from various locations and replay the errors in the simulation setup described above. Next, we average the TCP throughputs for all the trace-driven simulations and compare it with the throughputs of the error models.

7.2 Comparing accuracy of models

Table 5. TCP Throughput for Error Models

	Throughput (Mbps)	relative difference
Uniform	1.513	0.213
Markov	1.405	0.127
Empirical	1.307	0.048
Trace	1.247	0.000

Table 5 lists the average TCP throughputs for the three models and the trace modulation. Having the trace-driven simulation throughput as the reference, the relative difference is computed for each model. The empirical model leads to throughputs of about 5% higher than that of trace-driven simulations. The Markov and uniform error models differ by 13% and 21%, respectively. The higher TCP throughput for the Markov model suggests that the error lengths generated by the this model (shown in Figure 8) are shorter than the real ones.

8 Conclusion

We have shown that our trace-based approach accurately models wireless errors. A large number of traces that we have collected provide a good basis for developing and validating several wireless error models. From the trace data, we derive the parameters for two variations of the two-state error model. Although the Markov model is simpler, it leads to TCP throughputs that are, on average, 13% higher than those of the trace-driven simulations. The empirical model improves the accuracy to within 5%. Although this empirical model requires 13 parameters, it is not computationally expensive because the underlying statistical expressions are simple. The complexity comes mainly from the derivation of parameters. While the simple Markov model may be adequate for some applications, the higher accuracy of the empirical model will justify its complexity for others.

Although our goal is to provide the channel error models that are effective across many network protocols, the results obtained thus far are more suitable for evaluating higher-level network protocols. This is partly due to our immediate interest in TCP performance evaluations. We believe it is possible to improve this model for study link-level and MAC protocols by using smaller packet sizes in tracing.

9 Future work

Currently, we are applying our trace-based technique to realistically emulate wireless links with wired networks. Using the tools that we have developed for tracing, modeling, and simulation, we plan to investigate the interactions of different wireless devices operating in the same frequency band and their impact on transport protocols. With this knowledge, we can design better algorithms and protocols across several network layers to improve the overall performance.

Acknowledgments

I wish to thank Professor Randy Katz for advising me throughout this project. I also wish thank Professor Steven McCanne, Professor Mahadev Satyanarayanan, Brian Noble, Hari Balakrishnan, Bruce Mah, and Venkata Padmanabhan for many helpful suggestions in tracing and modeling of wireless networks.

References

[ARY95] Andersen, J. B., T. S. Rappaport and S. Yeshiva. 1995. Propagation Measurements and Models for Wireless Communications Channels. *IEEE Communications Magazine*, pp. 42-49.

[ATT93] AT&T Global Information Solutions Company. 1993. Architecture Specification for WaveLAN Air Interface.

[BB94] Bantz , D. F., and F. J. Bauchot. 1994. Wireless LAN Design Alternatives. *IEEE*, pp. 43-53.

[BB95] Bakre, A. and B. R. Badrinath. 1995. I-TCP: Indirect TCP for Mobile Hosts. *Proceedings of the 15th International Conference on Distributed Computing Systems*, pp. 136-143.

[BSK95] Balakrishnan, H., S. Seshan, and R. H. Katz. 1995. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *Wireless Networks*, pp. 469-481.

[C94] Chen, K-C. 1994. Medium Access Control of Wireless LANs for Mobile Computing. *IEEE Network*, pp. 50-63.

[CI95] Caceres, R. and L. Iftode. 1995. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE*, pp. 850-857.

[CL75] Cox, D. C. and R. P. Leck. 1975. Distributions of Multipath Delay Spread and Average Excess Delay for 910 MHz Urban Mobile Radio Paths. *IEEE Trans. Ant. Prop.*, vol. 23: pp. 206-213.

[CL94] Couture, F. J. and J.-P. Linnartz. 1994. Improved Channel Modeling for Performance Analysis of a DS-CDMA Link. *Proceedings of Wireless Networks Catching the Mobile Future*, vol.2: pp. 705-710.

[DBCF95] Davies, N., G. S. Blair, K. Cheverst, and A. Friday. 1995. A Network Emulator to Support the Development of Adaptive Applications. *Proceedings of the 2nd USENIX Symposium on Mobile and Location Independent Computing*.

[ES96] Eckhardt, D., and P. Steenkiste. 1996. Measurement and Analysis of the Error Characteristics of an In-building Wireless Network. *Proceeding of the ACM SIGCOMM*, pp. 243-254.

[FJ93] Floyd, S. and V. Jacobson. 1993. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, vol. 1,4: pp. 397-413.

[FJ95] Floyd, S. and V. Jacobson. 1995. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transaction on Networking*, vol. 3,4: pp. 365-386.

[F67] Fritchman, B. D. 1967. A Binary Channel Characterization Using Partitioned Markov Chains. *IEEE Transactions on Information Theory*, pp. 221-227.

[J91] Jain, R. 1991. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. *Wiley*.

[JS76] Kemeny, J. G. and J. L. Snell. 1976. Finite Markov Chains. *Springer-Verlag*, New York.

[KMM95] Kohno, R., R. Meidan, and L. B. Milstein. 1995. Spread Spectrum Access Methods for Wireless Communications. *IEEE Communication Magazine*, pp. 58-67.

[LMM95] Liu, G., A. Marlevi and G. Q. Maguire Jr.. 1995. A Mobile Virtual-distributed Architecture for Supporting Wireless Mobile Computing and Communications. *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking*, 111-118.

[MF95] McCanne, S., and S. Floyd. 1995. NS (Network Simulator). http://www-nrg.ee.lbl.gov/ns/

[MJ93] McCanne, S., and V. Jacobson. 1993. The BSD Packet Filter: a New Architecture for User-level Packet Capture. *Proceedings of the 1993 Winter USENIX Technical Conference*.

[MS94] L. Mummert and M. Satyanarayanan. 1994. Long Term Distributed File Reference Tracing: Implementation and Experience. CMU-CS-94-213.

[NNSK96] Noble, B., G. Nguyen, M. Satyanarayanan, R. H. Katz. 1996. Mobile Network Tracing. *RFC Draft*.

[SA+93] Schilit, B., N. Adams, R. Gold, M. Tso, and R. Want. The PARCTAB Mobile Computing System. *Proceedings of the 4th IEEE Workshop on Workstation Operating System*, pp. 34-39.

[WHFG92] Want, R. A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transaction on Information Systems*, vol. 10,1: pp. 91-102.

APPENDIX A

In designing our trace format, we have been inspired 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 forcing the trace format to be redesigned, we have structured it in an extensible way. There is a built-in mechanism to add to the kinds of data that can be recorded in the network traces.

The 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 the extended traces, if perhaps in a less than optimal way.

In our experience with other tracing systems, management of trace files can be difficult at best. Common problems include the need to manage multiple trace files as a unit, the ability to extract the salient features of large trace files, and the required use of dedicated trace management tools to perform even the simplest tasks. To alleviate the first two problems, we have made subsets of a trace file that should be considered a self-contained unit, allowing traces to be merged or split. To alleviate the last, we have chosen to store some descriptive information as ASCII strings, allowing minimal access to the standard tool suite.

This section defines the trace format that describes mobile environment and networking activity. The basic abstractions of the trace format are *records* and collections of records called *tracks*. A track is composed of a single header and many entry records. There are three kinds of tracks: *general track, packet track,* and *device track*. Packet tracks capture information from the packets that make up the workloads. Device tracks record device-specific information such as signal quality. All other types of data that does not fit into the packet and device category will be put into general tracks. Within a track, each record not only contains the required data fields, such as record time stamp, but also optional data fields that are both extensible and self-describing. These optional data are known as *properties* of the track. The following sections will provide detail description of the trace format, the required data fields and optional properties that have been defined to date. The description also includes special purpose trace records such as the trace headers, footer, annotation and data loss.

Records

A trace file is composed entirely of records that are contiguously stored in binary format. Binary representation is favored over ASCII for compactness, though there are a few exemptions that will be discussed in later sections. Due to the extensible requirement of the trace format, the record length is allowed to vary. The data fields that make up a record are stored in network byte-order if they are not byte addressable. Each field in a record is a four-byte word, which is defined to be **`u_int32_t`**. Although this is somewhat wasteful in space, it greatly simplifies the trace handling and allows the trace format to evolve gracefully.

Each record begins with a *magic* number, which is a fixed bit sequence that uniquely identifies the record type. The second field of each record specifies the size of the records in bytes. Together, these two fields lend confidence that the record is well formed since the size can be checked against known constraints, namely the size of the required and optional data fields for that record type. The remainder of the record is type-specific data, including the required and optional data fields for that particular record type.

Tracks

To make the trace format extensible, trace records are organized by tracks. Each track captures a set of observations about a particular entity such as all signal measurements for an NCR WaveLAN network interface or all TCP packets. Since the types of data vary from one entity to another, the records should have optional as well as required data fields. The optional fields can be compactly defined in the first record, called *track header*, of each track. Together, the definitions of optional data fields make up the *property list* of a track. Subsequent records belonging to the same track are called *track entries*.

```
struct tr_plist_ent_t {
   u_int8_t  pl_flags;         /* true if header-only */
   u_int8_t  pl_type;          /* type of the recorded data */
   u_int16_t pl_size;          /* size of the data in bytes */
   u_int32_t pl_name;          /* name of the property */
   u_int8_t *pl_value;         /* only for header property */
};
```

Figure 11. Property List Entry

The property list is an optional part of the track header. There are two types of property: *header-only* and *record-only*. A flag distinguishes these two types. Header-only properties contain values that hold true for the entire track. The remaining properties must have corresponding values in the optional data portion of the track entries. Each property entry in the header also defines its name, type, total size. The name is a unique identifier for the property. The type field specifies a basic data type that makes up the property value. The size field specifies the total number of bytes needed for the value field. The value field of record-only property has zero size and no meaning.

General Tracks

The most basic type of track is called the *general track*. Its header carries a minimum number of fields required for all track headers. Beside the magic number and size required for all records, a track header records the track identifier it is defining, its creation time, the IP address of the tracing host, and an optional property list. Each time stamp is composed of two words similar to the **`timeval`** structure in UNIX. The only difference is in the fraction-of-a-second field, which can be either microseconds or nanoseconds. With unit for this field is recorded in the trace header, all records in a trace file must have a uniform time structure.

```
struct tr_gen_hdr_t {
   u_int32_t gh_magic;
   u_int32_t gh_size;           /* size of this record in bytes */
   tr_time_t gh_timestamp;      /* track's creation time */
   u_int32_t gh_trackmagic;     /* magic # of the track entry */
   u_int32_t gh_ipaddr;         /* IP address of tracing host */
   tr_plist_ent_t *gh_plist;    /* variable property list */
};

struct tr_gen_ent_t {
   u_int32_t ge_magic;          /* TRACK identifier */
   u_int32_t ge_size;           /* size of this record in bytes */
   tr_time_t ge_timestamp;      /* when this record is generated */
   u_int32_t *ge_vlist;         /* optional data */
};
```

Figure 12. General Track Header and Entry

Following the simplicity of the general track header, the general track entry also keeps the required data fields to an absolute minimum. These fields must include the magic number, size, and time stamp. The rest are optional data. The minimum size for a track entry is 4 words or 16 bytes. Although the general track can record any data, there are two kinds of information are common enough to have their own track format. The following sections will describe the specialized tracks intended to capture packets and device information.

Packet Tracks

For both wire and wireless networks, capturing network activity is the most crucial function of the tracing facility. Most of the design effort for the trace format is focused on this single component to make sure it can trace the network with the highest possible accuracy. With the trace format described above, all network activity is captured in multiple packet tracks. The differences in networking devices and/or protocols serve as natural criteria to group the packets into tracks. These grouping of packets can be further focused on some subset of the total traffic handled by the machine.

Similar to the general track, each *packet track* starts with a header that identifies the tracing host as well as the starting time of that particular track. Since the grouping of packets into tracks are primarily based on the network interfaces and protocols, the packet track headers also record the network interface address and the protocol on which the packets are captured. These fields enable post-processing software to interpret device or protocol-specific options. The header concludes with the property list for the packet track.

Following the packet track headers are their corresponding packet entries. Beside the standard data fields made up of magic number, record size, and time stamp, each packet entry carry the packet size, **`pe_psize`**, which is the size of the packet in bytes.

Device Tracks

Device tracks are intended to capture the device-specific information that is important for studying wireless networks. The various signal quality parameters that are measured by the network interfaces may have significant correlation with the loss characteristics. Similar to tracing packets, measurements from multiple devices are split into multiple tracks. The track header contains the network address as well as the device type for a single interface. Since all the measurements are device-specific, they should be defined in the property list. The corresponding device entries have no required data field besides the standard track identifier, record size and time stamp.

Trace Header and Footer

The design of the trace format mandates the use of multiple tracks. Although the track header is capable of describing any set of related data, it should not record properties that are common for the entire trace. The *trace header* and *footer* are designed to capture this kind of information. All trace files begin with a trace header and end with a trace footer. Most properties of a trace can be obtained at the start time, therefore, should be included in the trace header. This reduces the purpose of the trace footer to mark the completion of the trace. If the footer is not the last record, the data that follows should be ignored and the trace file should be truncated. On the other hand, if the trace footer does not appear in the trace, the trace collection process might terminate prematurely or not complete. Dealing with these irregularities probably requires some input from the user.

Both the trace header and footer carry a time stamp indicating the start and end time of the trace respectively. The header also contains a variable-length trace description, which is an ASCII string that can provide details of how and under what conditions the trace is collected. Other useful pieces of information in the header include the host name and IP address of the machine that collected the trace. When multiple traces are merged, these fields may be changed depending on the contents of all the trace headers being combined. The details of the merge algorithm will be presented in later sections.