Tools and Techniques for Measurement of IEEE 802.11 Wireless Networks

Feng Li, Mingzhe Li, Rui Lu, Huahui Wu, Mark Claypool and Robert Kinicki
{lif,lmz,kkboy,flashine,claypool,rek}@cs.wpi.edu
CS Department at Worcester Polytechnic Institute
Worcester, MA, 01609, USA

Abstract— **With the growing popularity of wireless local area networks (WLANs) has come an increased need for effective measurements of real-world WLANs and their applications. This paper presents tools and techniques for measuring IEEE 802.11 WLANs. The techniques include details on setting up a PC as a wireless access point and building a wireless sniffer while the tools include programs for measuring link, network and application layer traffic. The tools are all open-source software available for download and the techniques all use open-source software and off-the-shelf hardware components. Together, these tools and techniques facilitate WLAN performance analysis across network layers in a flexible, accurate and cost-effective manner. To illustrate the usefulness of these tools and techniques for gathering WLAN measurements three case studies are presented: a streaming video session showing cross-layer performance; network characteristics of a wireless hand-held game; and measurements of access point queue size. Research employing these tools can yield more accurate WLAN models and more realistic evaluation of proposed WLAN changes in a network testbed.**

I. Introduction

The combination of the decrease in price of wireless local area networks (WLANs) and the increase in wireless link capacities has significantly increased the number of WLANs in homes, corporate enterprise networks, academic campus networks, and entire cities. Initially, much of the WLAN research was conducted primarily through the use of analytic models [2], [4] and simulation techniques [22], [7], [5]. Only recently have researchers tackled the task of measuring WLANs [14], [21], [24] to understand performance anomalies and the implications of installation choices.

However, accurate WLAN measurements have proven more elusive than measurements for wired LANs due to the unique characteristics of the wireless medium. For example, measurements over a single wireless hop, such as in an 802.11 infrastructure network, can vary depending upon the hop distance, cross and contending traffic, the building structure and even the human motion within a measurement testbed. Generally, capturing aspects of WLAN performance requires more than collecting measurement data at any one layer in the protocol stack. Instead, measurement tools and techniques that enable the researcher to observe the intertwined effects between network layers are important for a comprehensive assessment of WLAN performance [12].

Building network testbeds, a common approach used to run controlled experiments, provides special challenges within WLANs where accurate multi-layer measurements require custom hardware and software solutions and realistic measurements tend to come from commercial, often proprietary, black-box software and hardware. For example, wireless sniffers, while effective for trouble shooting and other diagnostic functions, are typically expensive and closed-source devices that offer less flexibility with respect to capturing specific performance metrics compared to employing open-source solutions. While inexpensive, today's commercial wireless access points (APs) remain as black-box components in the WLAN in that their exact internal configurations and protocol implementations are not normally known.

This paper presents useful tools and techniques for measuring IEEE 802.11 WLANs in real-world environments. One technique includes the details of modifying a PC to function as a host AP using off-the-shelf hardware and open-source software. Having a host AP facilitates the ability to gather measurements concurrently from several network layers and also provides the ability to experiment with customized changes within the AP. Another technique provides details on building a wireless sniffer to capture 802.11 and higher network layers. Having a wireless sniffer as a measurement tool enables non-intrusive network measurements and evaluation of proprietary 802.11 devices. Tools at the end-host allow measurement of link layer characteristics obtained through the wireless interface card and and IP layer metrics such as round-trip time and loss rate. Application-layer tools, previously developed for wired-network testbeds, are mentioned to provide information for complete wireless network experimentation. All the tools and techniques discussed in this paper are open-source software available for download at `http://perform.wpi.edu/tools/`.

This paper is organized as follows: Section II presents tools and techniques for IEEE 802.11 network measurement; Section III illustrates the use of the tools and techniques with three performance evaluation examples; and Section IV provided conclusions and future work.

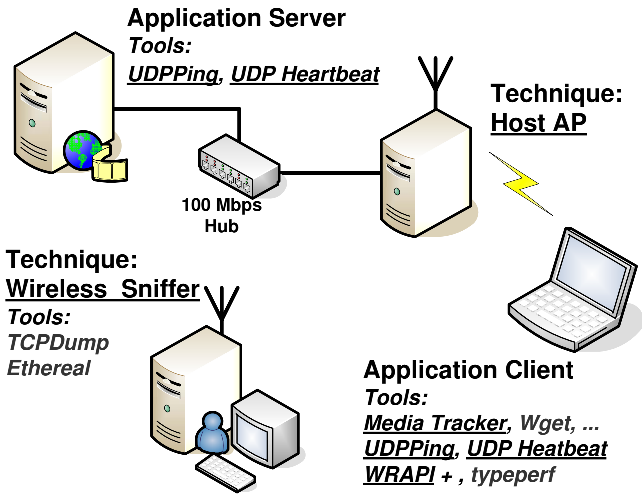

Fig. 1. Deployed tools and techniques. Underlined tools are discussed in this paper.

II. Techniques and Tools

Figure 1 depicts a basic testbed deploying the tools and techniques presented in this paper, with the underlined links being specific contributions of our work.

Baseline measurements of WRAPI+ and UDP Heartbeat indicated that these tools consume only about 3% of the client's processor time [18]. Measurements of the network sniffer reported no packets dropped due to being overloaded over a range of wireless traffic conditions [8]. Moreover, previous experience with application tools Media Tracker and Real Tracker indicate that they also add little overhead to application layer measurements.

A. Techniques

1) Build a Wireless Sniffer: While sniffers have been widely used to monitor network traffic at the data-link layer and above, most commercial wireless sniffers are costly and are not a flexible open source solution. However, passive sniffing does not interfere with the hosts under test and does not require access to the hosts themselves. Thus sniffers can be used to measure black-box devices such as hand-held game consoles (see Section III-B).

Our first wireless measurement technique is to turn a PC into an IEEE 802.11 wireless sniffer from open source software and off-the-shelf wireless networking hardware. Briefly, the steps include:[1]

1) Install the Linux operating system. The Linux distributions tested include SUSE (Novell) Linux releases 9.0, 9.1, 9.2 and 10.0 and Red Hat Fedora Core 3, with Linux kernel versions 2.4 and 2.6.
2) Integrate a prism GT-based wireless network interface card. The wireless network interface cards tested include Netgear WG 511 version 1 PCMCIA card and Allnet ALL0271 54 Mbit Wireless PCI adapter.
3) Update the driver (a prism54 kernel module) to the latest version.[2]
4) Create an interface configuration file that brings the wireless interface up in monitor (sniffing) mode.
5) Use network sniffing tools to capture frames. Popular network tools to capture and analyze capture data include tcpdump,[3] Ethereal,[4] or Kismet.[5]

The wireless sniffer captures the IEEE 802.11 Medium Acces Control (MAC) layer header, which includes data-link frame information such as the number of MAC layer retries, power management functionality and WEP encryption information. Additionally, the wireless sniffer collects extra physical and MAC layer information provided by the Absolute Value Systems (AVS) header,[6] an "extra" header which is created by the wireless driver when operating in monitor mode. With the AVS header, the wireless driver records wireless layer metadata, including the received signal strength, channel number and current capacity, and passes the metadata to the sniffer as an emulated frame header. Beyond performance monitoring, the sniffer can also capture wireless management frames such as RTS/CTS, (De)authentication, and (Dis)association. This extra functionality means the sniffer can be used as a wireless network diagnostic tool.

2) Build a Host Access Point: Results from prior WLAN measurement studies [1], [12], [18], [22] indicate that the internal allocation of buffers within a wireless AP can significantly impact the performance of WLAN applications. Moreover, these studies suggest that providing AP tuning mechanisms could yield performance gains. However, commercial APs are essentially "black-box" commodities that all promise equivalent performance regardless of internal design differences. This leaves academic researchers with little ability to evaluate and understand the internal workings of APs. Hence, the goal is an open source AP that runs on a PC – the host AP.

The first four steps in building a host AP are the same as the steps for building a wireless sniffer, presented in Section II-A.1. The additional steps are:[7]

5) Create an interface configuration file to bring up the wired interface.
6) Configure Dynamic Host Configuration Protocol (DHCP) to provide the addressing services

[1] See http://perform.wpi.edu/tools/ for details on each step.

[2] http://www.prism54.org/
[3] http://www.tcpdump.org/
[4] http://www.ethereal.com
[5] http://www.kismetwireless.net/index.shtml
[6] http://www.ethereal.com/docs/dfref/w/wlancap.html
[7] See http://perform.wpi.edu/tools/ for details on each step.

common to most APs. The DHCP[8] daemon (DHCPD) can be easily configured by modifying its configuration file based on a sample at `http://perform.wpi.edu/tools/`.

7) Share the connection to the wired and wireless adaptors. One of many possible solutions is to modify `iptables`[9] using a sample configuration file from `http://perform.wpi.edu/tools/`.

With the host AP built, an end-host wireless client using a commercial AP should be able to transparently associate and use the host AP, instead. Control of the internal components of a host AP allows exploration and understanding of the ramifications of internal AP resource allocation decisions on overall WLAN performance.

B. Tools

1) Wireless Layer: For monitoring WLAN performance on end hosts, a real-time monitor tool named *WRAPI+* was developed to extract information from the end hosts' IEEE 802.11b/g network device. WRAPI+ is an application built upon the freely available C++ library named WRAPI[10] that provides function calls to gather wireless statistics in Windows XP. The WRAPI Web site provides significant documentation on the features of WRAPI, but less information available for building applications that use WRAPI. For example, one initial appeal of WRAPI was the claim to be "a hardware-independent tool that works with any IEEE 802.11b wireless network hardware vendor". However, during the development of WRAPI+, hardware dependent problems were a major issue as several wireless adaptors were found to not be fully supported by the WRAPI library.

Thus, a contribution is WRAPI+, an application to monitor wireless statistics in real-time, with guidelines on the wireless adaptors supported (the PrismGT/2/2.5/3 chipsets). When run on an end-host, WRAPI+ periodically (every 500 ms, by default) reports and logs IEEE 802.11 network interface statistics that includes received signal strength, transmitted frame count, and failed frame transmissions and acknowledgments. WRAPI+ is provided with source code to maximize reuse and enable measurement customization.

2) Network Layer: Preliminary experiments revealed that the system `ping` (using ICMP) provided by Windows XP waits for the previous ping reply or a timeout before sending out the next ping packet. Hence, a constant ping rate cannot always be maintained under poor wireless conditions where long round-trip times can be encountered.

There are other previously developed tools to measure network layer performance, such as *Iperf*,[11] that measures maximum TCP and UDP bitrates, with reports for bandwidth, delay and loss, and *NetDyn*, [12] a low-overhead probing tool to measure network characteristics. However, one advantage to developing customized IP layer tools is this approach provides transparency as to the means by which network performance is inferred. Since such network layer code is fairly simple to write, a customized ping tool, *UDP Ping*, was built using application-layer UDP packets to provide configurable ping intervals and packet sizes. UDP Ping uses a server and a client. The UDP Ping client writes a sequence number and current timestamp into a UDP packet and sends it to the server which echoes the packet back to the client. The client then records round-trip time and is able to calculate the packet loss rate.

WLAN performance is often not symmetric, with network congestion in only one direction or with the upstream flows having more contention than the downstream flows. In these cases, UDP Ping can only report the aggregate network performance for both directions. To measure performance for each direction, a variant of UDP Ping named *UDP Heartbeat* was developed to report one-way delay and packet loss. The UDP Heartbeat sender writes a sequence number and current timestamp in the UDP packet and sends it to the receiver. Upon receiving the packet, the receiver calculates the time difference and reports any missing packets. To measure absolute one-way delay, the sender and receiver clocks must be synchronized. However, often only the relative difference in one-way delay time, say after network congestion starts, is needed to evaluate performance. Thus UDP Heartbeat can be a a useful measuremnt tool even when clocks are not synchronized.

3) Application Layer: At the application layer, there are numerous possible tools that drive network performance workloads and provide application-centric measures of network performance. For example, application measurement tools for streaming video such as *Media Tracker* [19] and *Real Tracker* [23], can provide a realistic video workload while provide application layer data specific to streaming video including: encoding data rate, playout bitrate, time spent buffering, video frame rate, video frames lost, video frames skipped, packets lost and packets recovered.

There are certainly other application layer tools worth considering, including *wget*,[13] a publicly-available HTTP/FTP download application that can be used to estimate the effective throughput of a TCP bulk transfer, and *httperf*,[14] a tool for measuring web server performance.

III. Case Studies

This section presents three case studies to demonstrate the use of the WLAN measurement techniques and tools discussed in Section II. The first case study shows cross-layer performance measurements for a video being streamed over a WLAN to a mobile user (Section III-A). The second case study utilizes a wireless sniffer to characterize the

[8] http://www.isc.org/products/DHCP/

[9] http://www.netfilter.org/

[10] http://sysnet.ucsd.edu/pawn/wrapi/

[11] http://dast.nlanr.net/Projects/Iperf/

[12] http://www.cs.umd.edu/˜suman/netdyn/index.html

[13] http://www.gnu.org/software/wget/wget.html

[14] http://www.hpl.hp.com/research/linux/httperf/

wireless network characteristics of two hand-held video games over a WLAN (Section III-B). The final case study employs a host AP to assist in understanding queues within an AP (Section III-C).

A. *Streaming Video over WLAN*

To show the importance of gathering wireless LAN measurements from multiple layers of the protocol and to provide a feel for how the tools we developed can be used in a complementary fashion, this section presents measurements from an experiment where a video is streamed from a Windows Media server to a mobile client over a WLAN. Specifically, this section shows wireless measurement results gathered simultaneously at the application, network and data link layers of the end-host.

A three-minute video clip of a moving Coast Guard ship was encoded with 11 bitrate layers and streamed over UDP from a Windows Media server, located on the wired part of the WPI campus, to a Media Tracker client on a laptop being carried by a graduate student volunteer. The volunteer was initially standing very close to a wireless AP. After approximately 10 seconds, the volunteer walked[15] slowly away from the AP.

Figure 2 offers a multi-layered snapshot of the data gathered. The top three graphs show application layer performance measurements from Media Tracker that include the video encoding bitrate, the received bandwidth, and the frame rate, respectively. The middle two graphs show network layer performance measurements from UDP ping that represent the round-trip time and the IP packet loss rate, respectively. The bottom three graphs show wireless layer performance measurements from WRAPI+ that include the target bandwidth level, the retransmission failures, and the signal strength, respectively.

Initially, the highest quality encoding layer, 2.4 Mbps, is selected for streaming by the media server. Consistent with results in [20], the client buffers from 2 seconds until about 18 seconds at a rate much higher than the video encoding rate, peaking around 3.75 Mbps. This suggests that the high playout rate overflows a downstream buffer from the server to the client and causes significant IP packet losses. This can be seen in the middle graph labeled "Loss Rate" (4th from the bottom) where there is a spike in loss rates up to 20% in the first 20 seconds, even though the wireless signal strength is still quite good.

Near the 10 second mark, video frame playout begins as seen by the graph labeled "Frame Rate" (3rd from the top). As the volunteer moves the client laptop away from the AP, the wireless signal strength steadily degrades, shown in the graph labeled "Signal Strength" at the bottom. However, the wireless target bandwidth remains high except for a brief, abrupt and unexplained change in bandwidth to 2 Mbps at 65

[15]Note, a mobile wireless client provides a challenging test for measurement tools over a wide range of network conditions.

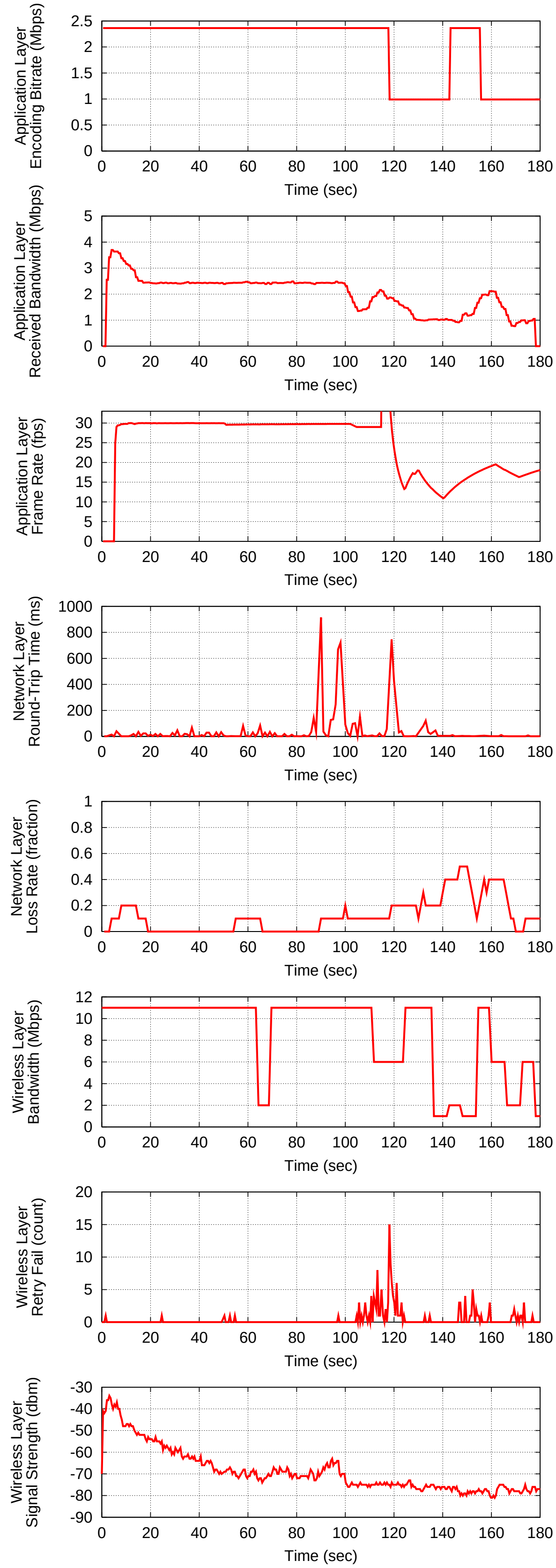

Fig. 2. Application layer, network layer, and wireless layer performance of a mobile client streaming a high-quality, multi-layer video clip.

seconds, seen in the graph labeled "Bandwidth" (3rd from the bottom).

At approximately 90 seconds, there is a sudden spike in round-trip time reported by UDP Ping that is accompanied by a considerable increase in IP packet losses, shown in the middle-two graphs labeled "Loss Rate". The streaming buffer built up at the client is able to smooth over these losses, keep the same encoding rate and maintain a consistently high frame rate.

Just before 100 seconds, the streaming wireless system encounters a major event: the round-trip time spikes again; the wireless signal strength degrades further; numerous wireless frame retransmissions are recorded by WRAPI+ (shown in the "Retry Fail" graph, 2nd from the bottom) and higher IP packet loss rates are seen due either to wireless layer frame drops or to AP buffer overflow.

With a delay in reaction time to this major event, the wireless data link layer lowers its target bandwidth to 6 Mbps (see the third graph from the bottom) and the streaming application changes encoding levels to 1 Mbps (see top graph) just before 120 seconds. Notice, however, that the wireless streaming system has reached a volatile state and the target bandwidth oscillates for the remainder of the test. Even the 1 Mbps encoding rate chosen for streaming is still too high for the effective wireless capacity. This yields numerous IP packet drops and subsequently a low-quality video frame rate. One can conjecture that at 142 seconds, the streaming system mistakenly switches to the original 2.4 Mbps encoding rate possibly due to erroneous capacity estimates from a packet-pair technique.

The use of these measurement tools suggest possible improvements to the streaming system. If the media server had better information on the effective bandwidth of the wireless channel, application throughput, round-trip time and packet loss rate, the server could choose a more effective video encoding layer from among the many choices it could have picked below 1 and 2.4 Mbps. Moreover, this brief analysis demonstrates the value of being able to collect wireless measurements concurrently at multiple layers of the protocol stack.

B. *Hand-Held Wireless Network Games*

To demonstrate the ability to non-intrusively gather wireless layer traffic from commercial devices for traffic characterization, this section presents measurements from an experiment where video games are played on a WLAN network with the newest commercial hand-held gaming platforms, each equipped with IEEE 802.11 wireless networking interfaces for multi-player gameplay.

The traffic generated by one host on a WLAN can have dramatic impact on the performance of other hosts on the WLAN [1], [13]. Specifically, when there is a WLAN host with weak wireless connectivity, and hence low WLAN capacity, the performance of all WLAN hosts is severely degraded. Moreover, some hand-helds are specifically designed to operate at low WLAN capacities to conserve power. These results are especially important since preliminary evidence suggests some game phases for hand-held games may have adverse affects on throughput for Internet applications sharing the same 802.11 wireless channel [8]. To adequately plan WLAN network infrastructures, it may be important for engineers to have knowledge of the network load caused by hand-held game traffic.

Three hand-held Nintendo DSes were used to play two different three-player games in separate sessions. Sniffing of the 802.11 traffic between the hand-helds was performed using a Dell Inspiron 8600 laptop, running in close proximity (10 feet) to the hand-helds, and with the hand-helds in close proximity with each other. The two games chosen were *GoldenEye: Rogue Agent* (a first-person shooter game) and *Super Mario* (a third-person action game).

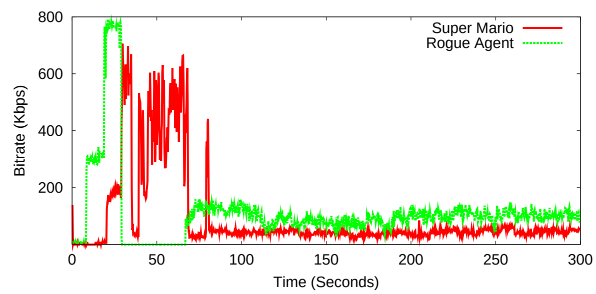

Fig. 3. Bitrate comparison for two Nintendo hand-held games.

Figure 3 shows bitrate (in Kbps) versus time (in seconds) for the two different games. Notice there are two distinct phases of network traffic in each game session. The first phase is marked by high bitrates where the hand-held devices send game setup data to each other as fast as possible. In this case, the data are the games themselves since the devices allow game sharing, but may also include game settings and map information. The setup phase is of different lengths for the two games. Rogue Agent ends its setup phase after about 25 seconds while Super Mario does not complete setup until just over 60 seconds.

The second phase, the play phase, is characterized by lower, more consistent data rates with bitrate quantities shown in Table I. During this phase where the game is actually played, players responds to the game state and the hand-helds communicate player actions to the other hand-helds, as appropriate. The bitrates during this phase are different for the two different games, with Rogue Agent having slightly higher bitrates (104 Kbps average, 15.6 std dev) than Super Mario (47 Kbps average, 9.1 std dev).

Since the play phase is normally significantly longer than the setup phase and is of the most concern to the players, a 50 second slice of the play phase for each game is analyzed to provide a uniform sample with which to compare the two

Game	Bitrate (Kbps)	Std dev
Super Mario	47	9.1
Rouge Agent	104	14.6

TABLE I

BITRATES BY GAME FOR PLAY PHASE (NOT INCLUDING SETUP PHASE)

games.

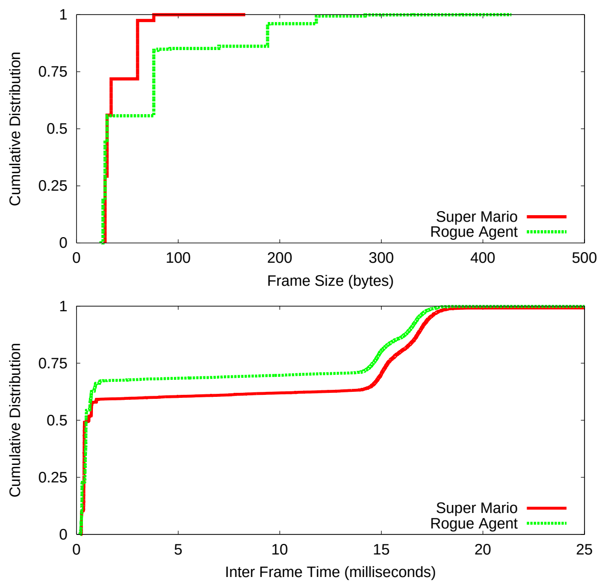

Fig. 4. Wireless frame analysis for two Nintendo hand-held games.

The top graph in Figure 4 provides the cumulative distribution functions (CDFs) for frame size while the CDF for inter frame time is presented in the bottom graph. In general, the frames are small and frequent, befitting the interactive nature of computer games. Compared with other Internet games, the hand-helds send frames considerably more frequently than a typical real-time strategy game, which has a signature of sending data every 100 ms [9], and more frequently than a typical first-person shooter game, which has a signature of sending data about every 40-50 ms [6], [25]. Wireless measurements such as bitrate, frame size and frame timings can be a useful first step in building models, such as in [25], [3], for hand-held game traffic.

Game	Management	Data	Overall
Super Mario	87	70	70
Rouge Agent	186	65	67

TABLE II

AVERAGE WIRELESS LAYER FRAME SIZE (BYTES)

Many of the frames are quite small, around 100 bytes (including all 802.11 headers) and many frames are sent within 1 millisecond of the previous frame. Most of these these small, closely-spaced frames are likely acknowledgments and account for approximately 50% of all frames recorded.

Game	Management frame (%)	Data frame(%)	Retry (count)	Total
Super Mario	1.75%	98.25%	17	62812
Rouge Agent	1.36%	98.64%	1	63965

TABLE III

WIRELESS LAYER STATISTICS

Table III facilitates further analysis of the wireless characteristics by showing there are few wireless frame retransmissions for either game (Rogue Agent had only 1 while Super Mario had 17). Moreover the two game sessions send roughly the same fraction of wireless management frames (beacon and authentication, and association frames).

For wired network games, traffic from a number of popular games has been characterized to provide suitable traffic models for testing existing or planned network designs. There have been numerous studies of traffic models for popular PC games [3], [6], [11], [17], [16] and even game consoles [25]. Use of a wireless sniffer can help collect data to build comparable models and for characterization of hand-held game traffic running over 802.11 networks.

C. Wireless Access Point Queue Behavior

Given the bursty nature of Internet traffic [15], wireless AP queue size can directly impact a flow's achievable throughput. While a small queue may keep effective bitrates significantly below the available capacity, a large AP queue size can negatively impact a flow's end-to-end delay which is detrimental to VoIP and some network game applications. Despite the importance of queue size to achievable throughput and delay, there is in practice little documentation on AP queue size settings.

To investigate queue sizes in wireless APs, a variant of the *QFind* [10] methodology is applied to infer the access network queue size for the host AP. UDP Heartbeat is run by itself for 15 seconds to provide a one-way baseline delay from a wired server, through the host AP to a wireless client. After obtaining a baseline, a high-bitrate UDP flow is sent concurrently with the UDP Heartbeat for 60 seconds to saturate the link and fill the host AP queue. The UDP Heartbeat records the increase in one-way delay between the baseline latency and the delay with the access queue filled by UDP load traffic. The host AP queue size can then be estimated by the product of the UDP flow throughput and the maximum delay increase measured by UDP Heartbeat. Let D_t be the total delay (the maximum delay seen by UDP Heartbeat):

$$D_t = D_l + D_q \tag{1}$$

where D_l is the latency (the baseline delay) and D_q is the queuing delay. Therefore:

$$D_q = D_t - D_l \quad (2)$$

Given throughput T (measured at the wireless client), the access link queue size in bytes, q_b, can be computed by:

$$q_b = D_q \times T \quad (3)$$

For packet size s (a 1400 byte application payload was used), the queue size in packets, q_p, becomes:

$$q_p = \frac{(D_t - D_l) \times T}{s} \quad (4)$$

The accuracy of the QFind method can be checked by instrumenting the Linux kernel in the host AP to monitor the transmit queue attached to the wireless adaptor. This instrumentation also illustrates the flexibility of an open-source AP. The instrumentation records queue size via a simple user-level application issuing a custom system call to obtain the queue size every 200 ms.

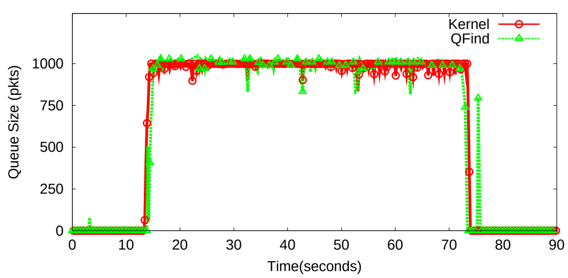

Fig. 5. Host AP with transmit queue length set to 1000 packets.

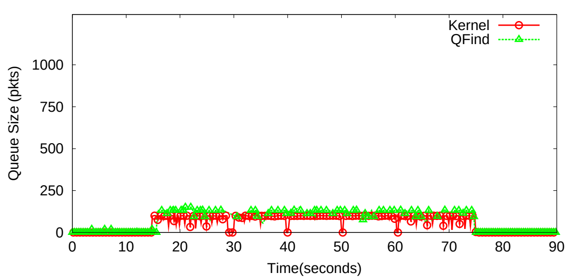

Fig. 6. Host AP with transmit queue length set to 100 packets.

QFind with kernel monitoring was run with the AP queue set to 1000 and then re-run with the AP queue set to 100. Figures 5 and 6, show the queue size (in packets) computed by QFind and reported by the host AP kernel on the y-axis as a function of time (in seconds) on the x-axis. Note that the *QFind* and *Kernel* data sets track each other very closely. This illustrates the efficacy of QFind as a methodology to determine queue sizes and suggests that if the host AP were replaced by a black-box, commercial AP, QFind could effectively determine the AP queue size.

Note that the Linux kernels v2.4 default queue size is 100 packets and Linux kernels v2.6 default queue size is 1000 packets. The differences between access queue sizes for different versions of Linux and perhaps for different commercial APs begs the question: what size *should* access queues be? The tools and methodology presented in this paper can be used to explore the impact of access queue size on throughput and round-trip times through wireless measurements. This in turn can be used to study the ability of wireless infrastructure topologies to provide quality of service to a variety of applications.

IV. Conclusions

This paper presents open-source software tools and techniques that facilitate exploration of the intertwined effects of WLAN performance across network layers in a flexible, accurate and cost-effective manner. Three examples are presented to illustrate the tools and techniques: 1) a streaming video session using applications, network and wireless layer tools aligns cross-layer performance; 2) wireless sniffing shows network bitrates, frame sizes and inter-frame times for wireless, hand-held games; and 3) data gathered with two queue size parameters demonstrates the use of a host AP. These tools and techniques are useful for gathering real data, both for building more accurate models of WLANs and their applications and for evaluating proposed improvements within a network testbed.

Future tool development such as incorporating new IEEE 802.11 standards (e.g., 802.11e and 802.11n), will allow exploration of state-of-the-art WLAN technology. Techniques and tools to measure and monitor 802.11 ad hoc mode, in particular ad hoc routing, and wire-area wireless technologies, can expand the research scope of the tools and techniques presented here.

REFERENCES

[1] G. Bai and C. Williamson. The Effects of Mobility on Wireless Media Streaming Performance. In *Proceedings of Wireless Networks and Emerging Technologies (WNET)*, pages 596–601, July 2004.

[2] G. Bianchi. Performance Analysis of the IEEE 802.11 Distributed Coordination Function. *IEEE Journal on Selected Areas in Communications, Wireless Series*, 18(3):535–547, Mar. 2000.

[3] M. S. Borella. Source Models of Network Game Traffic. *Elsevier Computer Communications*, 23(4):403 – 410, Feb. 2000.

[4] F. Calì, M. Conti, and E. Gregori. IEEE 802.11 Wireless LAN: Capacity Analysis and Protocol Enahncement. In *Proceedings of IEEE INFOCOM*, pages 142–149, San Francisco, CA, USA, Mar. 1998.

[5] M. Carvalho and J. Garcia-Luna-Aceves. Delay Analysis of IEEE 802.11 in Single-Hop Networks. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, Atlanta, Georgia, USA, Nov. 2003.

[6] W. chang Feng, F. Chang, W. chi Feng, and J. Walpole. Provisioning On-line Games: A Traffic Analysis of a Busy Counter-Strike Server. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop (IMW)*, Nov. 2002.

[7] P. Chatzimisios, A. C. Boucouvalas, and V. Vitsas. Performance Analysis of IEEE 802.11 DCF in Presence of Transmission Errors. In *IEEE International Conference on Communications (ICC)*, volume 27, pages 3854 – 3858, June 2004.

[8] M. Claypool. On the 802.11 Turbulence of Nintendo DS and Sony PSP Hand-held Network Games. In *Proceedings of the 4th ACM Network and System Support for Games (NetGames)*, Oct. 2005.

[9] M. Claypool. The Effect of Latency on User Performance in Real-Time Strategy Games. *Elsevier Computer Networks, Special Issue on Networking Issues in Entertainment Computing*, 49(1):52–70, Sept. 2005.

[10] M. Claypool, R. Kinicki, M. Li, J. Nichols, and H. Wu. Inferring Queue Sizes in Access Networks by Active Measurement. In *Proceedings of the 5th Passive and Active Measurement Workshop (PAM)*, Apr. 2004.

[11] J. Faerber. Network Game Traffic Modelling. In *Workshop on Network and System Support for Games*, Apr. 2002.

[12] J. Gretarsson, F. Li, M. Li, A. Samant, H. Wu, M. Claypool, and R. Kinicki. Performance Analysis of the Intertwined Effects between Network Layers for 802.11g Transmissions. In *In Proceedings of the 1st ACM Workshop on Wireless Multimedia Networking and Performance Modeling (WMuNeP)*, Oct. 2005.

[13] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance Anomaly of 802.11b. In *Proceedings of IEEE INFOCOM*, 2003.

[14] A. P. Jardosh, K. N. Ramachandran, K. C. Almeroth, and E. M. Belding-Royer. Understanding Congestion in IEEE 802.11b Wireless Networks. In *Proceedings of the Internet Measurement Conference (IMC)*, Berkeley, CA, USA, Oct. 2005.

[15] H. Jiang and C. Dovrolis. Source-Level IP Packet Bursts: Causes and Effects. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, Miami, FL, USA, Oct. 2003.

[16] T. Lang, G. Armitage, P. Branch, and H.-Y. Choo. A Synthetic Traffic Model for Half Life. In *Australian Telecommunications Networks & Applications Conference (ATNAC)*, Melbourne, Australia, Dec. 2003.

[17] T. Lang, P. Branch, and G. Armitage. A Synthetic Traffic Model for Quake 3. In *ACM SIGCHI Advances in Computer Entertainment (ACE)*, Singapore, June 2004.

[18] F. Li, J. Chung, M. Li, H. Wu, M. Claypool, and R. Kinicki. Application, Network and Link Layer Measurements of Streaming Video over a Wireless Campus Network. In *Proceedings of the 6th Passive and Active Measurement Workshop (PAM)*, Boston, Massachusetts, USA, Apr. 2005.

[19] M. Li, M. Claypool, and R. Kinicki. MediaPlayer versus RealPlayer – A Comparison of Network Turbulence. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop (IMW)*, pages 131 – 136, Marseille, France, Nov. 2002.

[20] J. Nichols, M. Claypool, R. Kinicki, and M. Li. Measurements of the Congestion Responsiveness of Windows Streaming Media. In *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2004.

[21] E. Pelletta and H. Velayos. Performance Measurements of the Saturation Throughput in IEEE 802.11 Access Points. In *In Proceedings of the Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pages 129–138, Apr. 2005.

[22] S. Pilosof, R. Ramjee, D. Raz, Y. Shavitt, and P. Sinha. Understanding tcp fairness over wireless lan. In *INFOCOM*, 2003.

[23] Y. Wang and M. Claypool. RealTracer - Tools for Measuring the Performance of RealVideo on the Internet. *Kluwer Multimedia Tools and Applications*, 27(3), Dec. 2005.

[24] M. Yarvis, K. Papagiannaki, and W. S. Conner. Characterization of 802.11 Wireless Networks in the Home. In *Proceedings of 1st workshop on Wireless Network Measurements (WiNMee)*, Riva del Garda, Italy, Apr. 2005.

[25] S. Zander and G. Armitage. A Traffic Model for the Xbox Game Halo 2. In *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Stevenson, WA, USA, June 2005.