Using Physical Layer Emulation to Optimize and Evaluate Mobile and Wireless Systems

Glenn Judd
Carnegie Mellon University
Pittsburgh, PA, USA
glennj@cs.cmu.edu

Xiaohui Wang
Carnegie Mellon University
Pittsburgh, PA, USA
xiaohuiw@ece.cmu.edu

Mei-Hsuan Lu
Carnegie Mellon University
Pittsburgh, PA, USA
meihsual@ece.cmu.edu

Peter Steenkiste
Carnegie Mellon University
Pittsburgh, PA, USA
prs@cs.cmu.edu

ABSTRACT

Testing and evaluating protocols and applications for wireless networks and mobile users is challenging because the physical environment has a significant impact on the behavior and dynamics of the system. It is however important that these physical world effects are considered during system implementation and evaluation to ensure correct and efficient operation. Unfortunately, since these physical world effects are hard to control and model, this adds considerable complexity to system development. In this paper we show how a wireless networking testbed based on signal propagation emulation was used in the development, testing, and evaluation of mobile systems. The paper is organized as two case studies at different levels of the system: roaming in 802.11 networks and video streaming. We found that the combination of realism and control improved both efficiency and performance during development.

Categories and Subject Descriptors

C.2 [**Computer Systems Organization**]: Computer-Communication Networks ; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design
Wireless communication

General Terms

Measurement, Performance

Keywords

Roaming, video streaming

This research was funded in part by NSF under award numbers CCR-0205266 and CNS-0434824. Additional support was provided by Intel and Xilinx.

MobiQuitous 2008, July 21 - 25, 2008, Dublin, Ireland.

1. INTRODUCTION

Testing and evaluating protocols and applications for wireless and mobile users is challenging because the physical environment has a significant impact on the behavior and dynamics of the system. For example, the physical infrastructure and movement by people and objects can create shadows and fading that create dynamic wireless conditions. Mobility of the wireless devices adds an additional dimension to channel dynamics. These physical world effects must however be considered during system development and evaluation to ensure correct and efficient operation. For this reason, developers typically test their code in variety of real world environments, but the lack of control over the physical world (movement by people, production networks, etc.) means that the experiments are not repeatable. This makes testing, debugging, and side-by-side comparison of different implementations challenging. Moreover, it may be hard to interpret results and the precise reasons for poor performance may be difficult to isolate

In response, developers have looked at alternative platforms. The obvious choice is simulators such as ns-2 [**?**] and OPNET [**?**]. They are widely used in the research community but it has been observed that the lack of physical layer accuracy can lead to the incorrect conclusions [**?**]. Moreover, simulator code does not always port easily to real systems. Researchers have also developed other techniques such as packet level emulation [**?**], analog channel emulation [**?**], partially controllable wireless testbeds [**?**, **?**, **?**], or hybrid solutions [**?**]. While these approaches are useful for certain classes of experiments, they also have limitations such as limited physical layer control or repeatability, limited flexibility, or high complexity.

In this paper we explore how a wireless networking testbed based on signal propagation emulation can be used in the development, testing, and evaluation of wireless and mobile systems. The testbed achieves a high level of realism since it uses real wireless devices but it also provides full control and repeatability at the physical layer. This paper is organized as a set of two case studies that focus on different levels of the system: the impact of roaming in 802.11 networks (MAC layer), and video streaming over wireless (application layer). Both roaming and video are very sensitive to timing, which in turn depends on the properties and dynamics of the wireless channel. Results on how the emulator supports

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3666
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3666

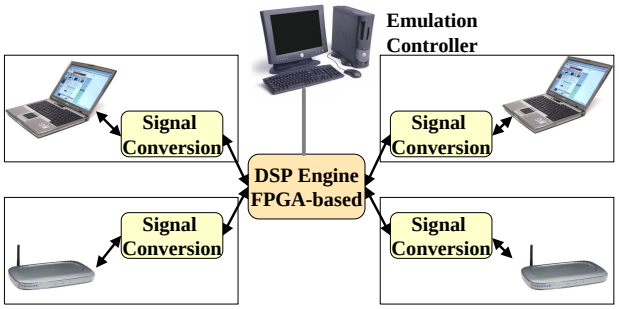


Figure 1: Emulator Implementation

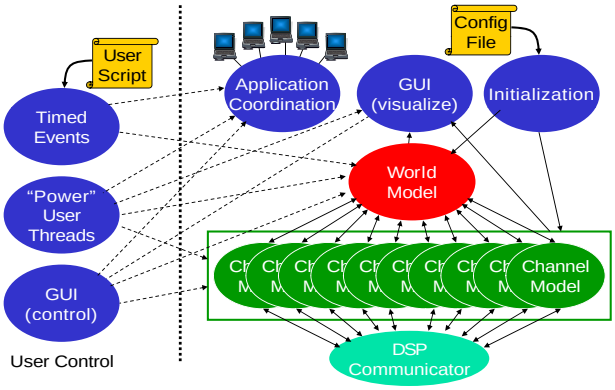


Figure 2: Emulator control software

physical layer studies have appeared elsewhere [?, ?].

The primary contribution of this paper is that we show how digital signal propagation emulation can improve the development, testing, and evaluation process of wireless and mobile systems. The control provided by a wireless emulation testbed can improve measurement accuracy and user efficiency, thus speeding up the development process. Other contributions include a novel approach to fast roaming for mobile users and an in depth evaluation of time-aware retransmission in 802.11.

The remainder of this paper is organized as follows. In the next section we give an overview of the wireless network emulator testbed. We present our two case studies in Sections 3 and 4. Finally, we summarize our results in Section 5.

2. WIRELESS NETWORK EMULATOR

Signal propagation emulation [?, ?] allows us to conduct network experiments using real wireless devices running in real-time in a controlled environment. The operation of our emulator is illustrated in Figure 1. A number of "RF nodes" (e.g. laptops) are connected to the emulator through a cable attached to the antenna port of their wireless networks cards. On transmit, the RF signal from a given RF node is passed into the signal conversion module where it is shifted down to a lower frequency, digitized, and then forwarded in digital form into a central DSP Engine that is built around an FPGA. The DSP Engine models the effects of signal propagation (e.g. large-scale attenuation, multi-path, and small-scale fading) on each signal path. Finally, for each RF node, the DSP combines the processed input signals from all the other RF nodes and sends it to the signal conversion module. It converts the digital signal back into a radio signal and sends it to the wireless line card through the antenna port. Our implementation supports the full 2.4 GHz ISM band.

The emulator simultaneously offers a high degree of realism and control. The RF nodes are shielded from each other (boxes in Figure 1) so that **no communication occurs over the air**. Since RF nodes only communicate through the emulator, we have full control over the signal propagation environment. The only simulated element is the *propagation* of signals between hosts. Channels are modeled at the signal level but the wireless hardware, signal generation, signal reception, and software on the end hosts are all real.

Emulation is controlled by an Emulation Controller executing on the Emulation Controller PC. The Emulation Controller models the emulated physical environment including the movement of the wireless devices (World Model in Figure 2). It also coordinates the movement of devices with the modeling of the signal propagation in the FPGAs by modifying its parameters in real time (Channel Models). For example, as nodes move in the emulated physical environment, the emulation controller adjusts the parameters for the large-scale attenuation and fine grain fading in a way that is consistent with the speed and location of the devices and writes the new values to the FPGA. The Emulation Controller can also control application behavior on the end hosts (Application Coordination). Each RF node runs a small daemon that allows the Emulation Controller to control its operation via a wired control network.

Users can specify and control wireless experiments in three different ways [?] (left side of Figure 2):

- *Interactive GUI* - Users can control key features of the experiments such as node placement and channel parameters using an GUI. When using the GUI, users often log into the laptops to manually control applications and monitor performance.
- *Scripting* - Users can use a scripting language to control nodes, applications and wireless channels. The scripting language is very simple, e.g. there is no control flow.
- *Programmatic control* - Users can use Java code to access to all emulator features by directly calling the appropriate classes. They can also replace or bypass modules, such as the "World Model".

3. OPTIMIZING ROAMING IN 802.11

Access point selection is a fundamental issue that has a large impact on wireless LAN performance, especially for mobile users. In 802.11, access points provide mechanisms for client association and disassociation, but clients are left to discover access points and implement policies that provide good performance. In this section we discuss how we used the wireless emulator testbed to develop a new fast roaming algorithm.

3.1 Roaming Challenges and Background

The first step in roaming is access point discovery, which is a slow process because 802.11 clients are half-duplex devices that operate on a single channel. As a results, communicating using a specific access point and looking for new ac-

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3666
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3666

cess points (which may be on other channels) are largely mutually exclusive operations. Let us briefly review the two access point discovery mechanisms provided by 802.11. With *passive discovery*, the client simply listens for a certain amount of time on each channel for incoming beacon frames that access points transmit periodically. The disadvantage of this approach is the latency involved. Beacons are typically sent out at 100 ms intervals, so a station must listen on each channel for at least 100 ms to hear from all access points. Thus scanning eleven 802.11b/g channels and/or 802.11a channels requires on the order of one to two seconds. The latency of the discovery process can be reduced by using *active access point discovery*. With active discovery, a client sends out a probe frame on a channel and then waits for for probe responses from the access points within range. While significantly faster than passive scanning, this process can still take several hundred milliseconds. Moreover, the recent trend towards centralized access point controllers could increase this latency in the future.

Once a client has a list of access points within range, it must decide what access point to associate with. Access point selection policy is not specified in the 802.11 standard, but given the paucity of information available to clients, the most realistic policy is to select the access point with the strongest signal. We will use this common policy in this work. The client can then associate with the access point. Several other functions may need to be performed, including authentication, acquiring an IP address, etc. The cost of these functions depends heavily on the context and it is often possible to reduce their overhead using caching, i.e. the cost is only paid once when associating for the first time with an access point in the enterprise [**?**]. For this reason we will ignore these costs and focus on access point discovery.

Channel scanning for a new access point can be triggered by a high loss rate of beacons from the current access point. Alternatively, roaming clients may want to proactively look for better access points. Client can avoid packet loss while scanning other channels by using power save mode. The client first sends a power safe mode message to its current access point. If there are no buffered packets in the access point, the client switches to a different channels, probes for new access points, waits for responses and switches back to its original channel. It can then collect any packets that may have arrived from the access point thus avoiding packet loss. Whenever the client discovers an access points with stronger signals, it disassociates with the old access point and associates with the new one.

3.2 Measuring Access Point Selection Performance

To better understand the performance of access point discovery and roaming, we designed two tests that examine a single roam operation between two access points in a very simple environment, where we can easily determine the performance of a "perfect" client. These simple emulator experiments provide insights into the protocol behavior that would be very difficult to obtain using other techniques. In this section, we describe the two tests and report and analyze the results obtained for the Madwifi driver.

3.2.1 Abrupt Roam Test

In the abrupt roam test, the client initially has a superb connection with access point A and no connectivity with access point B. At a certain time, t_{move}, the client is instantaneously moved from its location near access point A to a point near access point B. At that time, the connections are reversed so that it has no connectivity with A and excellent connectivity with B. We specified the experiment using emulator's programmatic control interface. This allows tight control over client movement and coordination of the measurement of access point selection behavior with the movement. We directly control path loss and do not use fine grain fading.

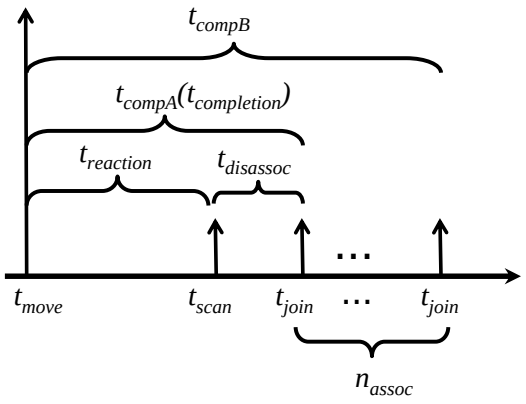


Figure 3: Roaming Metrics

We instrumented the client client driver to monitor the time t_{scan} when a non-background channel scan is triggered, as well as the times, t_{join}, when the client associates with a different station (access points B) after completing the scan. We also had a user process periodically capture the association status of the network interface using iwconfig. Using this information, we then calculated the following metrics (Figure 3):

- $t_{reaction} = t_{scan} - t_{move}$: the amount of time after t_{move} until the client starts a scan.
- $t_{completion} = max\{t_{join}\} - t_{move}$: the amount of time after t_{move} until the client is associated with access point B and no further associations or disassociations are observed.
- $t_{disassoc} = t_{join} - t_{scan}$: the amount of time during which the client was disassociated from all access points. In the abrupt roaming scenario, the client is disconnected longer than $t_{disassoc}$ because the client was not aware of the abrupt change until t_{scan} and it was technically still associated with access point A, although it can longer reach it. The client is really disconnected until $t_{completion}$.
- n_{assoc}: the number of access point associations. E.g. a client that disassociated with A, associated with B, disassociated with B, then associated with A would yield $n_{assoc} = 2$.

The "optimal" client behavior in this scenario is that the client should change access point as soon as possible, so the above times should be as small as possible.

The abrupt test was used to analyze the roaming performance of the Madwifi-NG driver, version 0.9.1, which was very popular at the time of this work. The wireless cards used were Senao NL-5354MP ARIES2 cards which are based on the Atheros 5212. Table 1 shows the results for a series of

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3666
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3666

	$t_{reaction}$	$t_{completion}$	$t_{disassoc}$	n_{assoc}
Mean	11.61 s	15.55 s	3.94 s	1.25
Median	7.69 s	12.40 s	2.14 s	1
Min.	5.09 s	7.20 s	2.07 s	1
Max.	23.18 s	27.58 s	9.24 s	2
Std.Dev.	6.91 s	7.82 s	2.71 s	0.44

Table 1: Abrupt roaming - Madwifi 0.9.1

	$t_{reaction}$	$t_{completion}$	$t_{disassoc}$	n_{assoc}
Mean	2.43 s	4.89 s	2.46 s	1.1
Median	2.00 s	4.17 s	2.16 s	1
Min.	0.94 s	4.05 s	2.13 s	1
Max.	11.53 s	14.66 s	4.31 s	2
Std.Dev.	2.16 s	2.36 s	0.69 s	0.31

Table 2: Abrupt roaming - Madwifi 0.9.1-E

	$t_{reaction}$	t_{compA}	t_{compB}	$t_{disassoc}$	n_{assoc}
Mean	13.92 s	16.05 s	16.05 s	0 s	1
Median	13.76 s	16.53 s	16.53 s	0 s	1
Min.	11.11 s	11.11 s	11.11 s	0 s	1
Max.	16.69 s	18.97 s	18.97 s	0 s	1
Std.Dev.	2.35 s	2.61 s	2.61 s	0 s	0

Table 3: Gradual roaming - Madwifi 0.9.1-E

20 runs of the abrupt test for the Madwifi 0.9.1 driver. The results clearly reveal two serious shortcomings. First, the driver is extremely slow to react to the new signal environment. It took over 7 seconds on average to begin searching for a new access point. Second, once the driver has recognized the need to find a new access point, it is quite slow in doing so. It took an average of another 5 seconds to find and associate with the new access point. During the roaming process, the client is either associated with an access point it cannot reach or it is not associated with any access point, so the client is disconnected from the network for over 12 seconds.

3.2.2 Gradual Roam Test

To observe behavior under a more typical roam scenario - but one in which the "optimal" outcome is still known - we designed the gradual roam test. The gradual test models the scenario of a person walking quickly between the two access points so that the change in signal conditions is gradual. Moreover, the client is never completely out of range of either access point. In the scenario we evaluated, access points A and B are 100 meters away from each other. We use a log distance large scale path loss model with a d_0 of 1 m, a pl_{d0} of 40 dB, and a path loss exponent n of 3. Again, no fading was used. The trip between the two access points takes 20 seconds, so the client is moving quite quickly. We picked this high rate of movement to stress the roaming support in the driver. The client then remains stationary near access point B for 20 more seconds before the test is repeated in the opposite direction. This process was repeated 20 times. When the client is close to an access point, e.g. B, it can communicate with that access point at 11 Mbps, achieving the maximum throughput of about 6 Mbps. At that time, connectivity with the access point A is very weak: the received signal strength from access point A is -92dBm and an iperf test shows a throughput of 1.5 Mbps. In this test, the optimal behavior is that the client switches access points when it is halfway between the two access points, i.e. after 10 seconds.

Under this test, the Madwifi 0.9.1 driver failed to roam despite very poor connectivity with access point A when located near access point B. For the entire test, the client remained associated with access point A. This test clearly shows that Madwifi 0.9.1's slow reaction to a changing signal environment performs very poorly in a mobile scenario.

3.3 Improving and Tuning Madwifi 0.9.1

The emulator's interactive GUI provided an ideal environment in which to investigate the causes of the poor roaming performance. We used the signal environment from the abrupt roam case, but we turned off the mobility script and instead controlled the node placement using the GUI. We logged into the laptops to enable roam debug logging and recorded the time of the association and disassociation results. By analyzing the results and looking through the code we were able to quickly identify several mechanisms that were contributing to the poor roaming performance. First, proactive scanning code was disabled so scanning was purely failure driven, i.e. the connection with the current access point had to break before scanning was initiated. Second, the signal strength results of a scan were sent through a low-pass filter, which caused the results to be updated very slowly. Thus, several scans were required to obtain an accurate view of the signal environment. Third, scan entries were cached. This may be appropriate for nomadic users, i.e. users who use their laptops in different locations, but it can result in the use of stale information for mobile users. For example, based on stale cache entries, multiple attempts were made to associate with access points that were not reachable. Finally, we also observed that the scanning algorithm is very slow and uses poorly tuned parameters.

3.4 Enhanced Roaming

We developed an enhanced version of Madwifi 0.9.1 - called Madwifi 0.9.1-E. - that addressing the problems identified above. We made three changes to the original code: (1) we enabled proactive scanning, (2) we disabled the low-pass filter of signal strength results, and (3) we disabled scan caching. We then repeated the abrupt and gradual roaming tests; the results are shown in Tables 2 and 3. We also repeated the gradual roaming test with Ricean fading (with $k = 3$), which is more realistic for a mobile scenario. Table 4 shows that the result are similar to those for the non-fading scenario.

The results show that in the abrupt roaming case, reaction and completion times have dropped significantly. In the gradual roaming test, in contrast to the original driver, the Madwifi driver now roams. In fact, in the tests we reduced the dwell time to 10 seconds as roaming clearly happens well under that. In the gradual roam case we need different definitions for $t_{completion}$ - labeled t_{compA} and t_{compB}. The reason is that the client may switch between access points multiple times during the test. Moreover, since proactive scanning is used, the client may be able to switch to the new access point before performance with the old access point has degraded, thus avoiding triggering an active scanning as in abrupt roaming. As a result, the definition of $t_{reaction}$ also need a slight modifications. The metrics for gradual roaming are defined as follows:

- t_{compA}: the amount of time after t_{move} until the client

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3666
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3666

	$t_{reaction}$	t_{compA}	t_{compB}	$t_{disassoc}$	n_{assoc}
Mean	13.15 s	16.14 s	16.55 s	0.41 s	1.3
Median	13.44 s	16.34 s	17.25 s	0 s	1
Min.	10.05 s	13.15 s	13.15 s	0 s	1
Max.	15.76 s	18.92 s	18.92 s	3.11 s	2
Std.Dev.	2.03 s	1.76 s	1.86 s	1.00 s	0.48

Table 4: Gradual roaming + fading -Madwifi 0.9.1-E

is first associated with access point B.

- t_{compB}: the amount of time after t_{move} until the client is associated with access point B and no further associations or disassociations are observed.
- $t_{reaction} = min\{t_{join}, t_{scan}\}$: the amount of time after t_{move} until the client either starts a scan or associates with the new access point.

Ideally in this case, roaming would occur halfway between the access points or at 10 seconds. The enhanced version performs within a couple seconds of this optimal time, but there is still room for improvement.

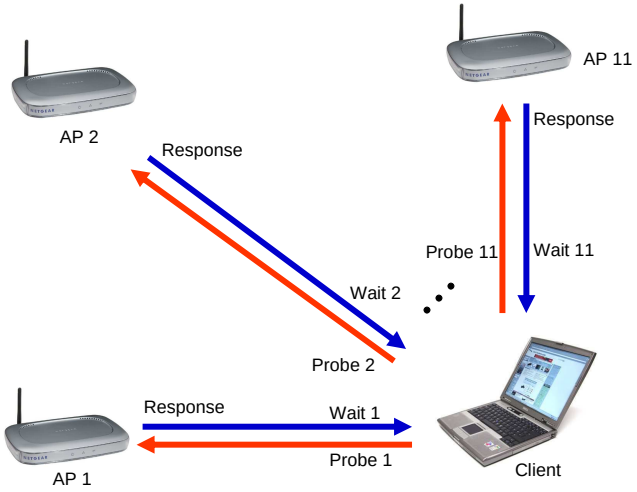


Figure 4: 802.11 Scanning

3.5 Access Point Discovery Using Fast Scanning

Our measurements of the Madwifi 0.9.1-E scan performance revealed that a scan operation typically takes around three seconds. To reduce this delay, we develop a fast scanning algorithm. Let us first review the operation of a traditional 802.11 active scan operation (Figure 4). When searching for access points, a scanning client scans all eleven channels by changing to the desired frequency, sending a probe request, and then waiting for probe responses. While a station is scanning, it cannot communicate with its access point. The scanning time is dominated by the need to wait for probe responses from access points. If no access point is heard on a given channel, the maximum channel dwell time must be used. For Madwifi 0.9.1 this is 200 ms. We can reduce the dwell time, but this only helps to a limited degree and it could cause missed probe responses.

Our fast scanning approach, shown in Figure 5, eliminates the time required to listen for probe responses. The

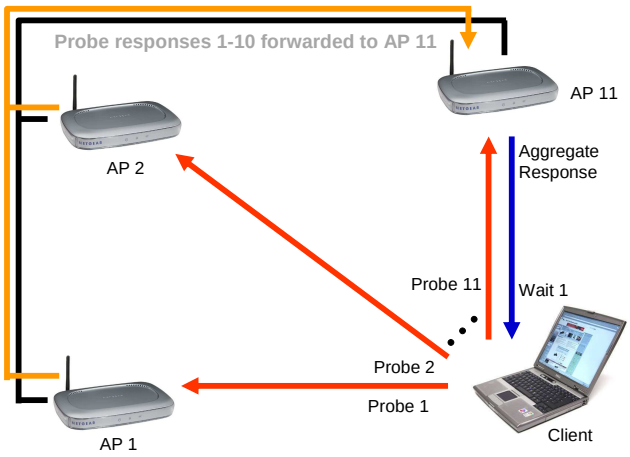


Figure 5: Fast Scanning

	$t_{reaction}$	t_{compA}	t_{compB}	$t_{disassoc}$	n_{assoc}
Mean	9.77 s	13.01 s	15.20 s	2.19 s	1.54
Median	9.46 s	11.89 s	12.53 s	3.07 s	2
Min.	6.29 s	8.54 s	11.52 s	0.00 s	1
Max.	14.70 s	19.02 s	20.09 s	5.33 s	2
Std.Dev.	3.20 s	3.35 s	3.79 s	2.27	0.52

Table 5: Gradual roaming - Madwifi 0.9.1-EFS

key idea is to have access points use the wired network to forward scan information to the scanning client's current access point, which then forwards it to the client. This means that the client only needs to wait for probe responses. As before, a fast scanning client visits all channels to send a probe request, but it then moves to the next channel without waiting for a probe response. The last channel scanned is the channel of its current access point. The access point collects the probe responses it receives from the other access points and sends an aggregate probe response to the client. This approach greatly reduces the amount of time required to complete a scan. In fact, all channels can often be scanned in less time than two channels can be scanned without fast scanning. Moreover, the only wait for probe reponses occurs when on the channel of the currently associated access point, so communication can still take place while waiting for probe responses.

We implemented fast scanning in the Madwifi 0.9.1-E driver. In the Madwifi 0.9.1-EFS driver, the client proactively scans every 500 msec and it uses power save mode while scanning to prevent packet loss, as described in Section 3.1. Table 5 shows the performance of the gradual roam test with Madwifi 0.9.1-EFS; again, the results shown are from 20 repetitions. The reduced scanning time enabled by fast scanning brings the scan completion time much closer to the optimal 10.0 seconds compared to Madwifi 0.9.1-E. Thus fast scanning appears to be a promising and viable approach. Results with fading are similar (table omitted).

The abrupt roam test performance was essentially the same as Madwifi 0.9.1-E. This was expected since since fast scanning was not implemented for the not-associated case, e.g. when a client initially joins a network. Fast scanning must be modified in that case since the client is not associated with any access point. One possible solution is to

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3666
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3666

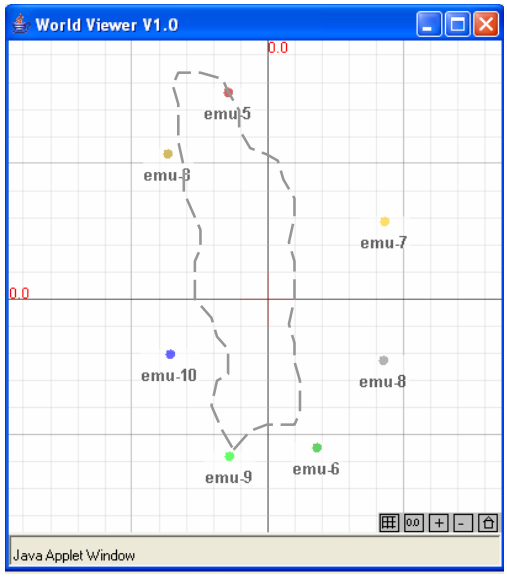


Figure 6: Multi-AP Test Topology

have the client associate with the first access point it encounters and then switch to fast scanning to find the best access point. We did not implement this since our focus is on optimizing roaming.

3.6 Multiple Access Point Performance.

In this section we evaluate the three Madwifi roaming variants using a more realistic scenario that involves a client roaming in an area covered by multiple access points. The goal is to maintain high throughput while avoid disassociation. We consider an application scenarios involving VoIP-like UDP traffic. The multiple-access point high-mobility scenario is shown in Figure 6. It consists of six access points (emu-3, emu-6, emu-7, emu-8, emu-9, and emu-10) that are distributed in the test area. They are set to channels 8, 10, 6, 4, 2, and 10 respectively. The client (emu-5) navigates a route that goes past the access points at a speed of 2 meters per second in a counter-clockwise fashion. A log distance large scale path loss model is used with a d_0 of 1 m, a pl_{d0} of 40 dB, and a path loss exponent n of 4.1. Ricean fading with $k = 3$ was used. Note that this is a very challenging scenario: the client has very poor connectivity to all access points during several parts of the route. This experiment was implemented using Java to give us good control over timing.

UDP-unicast traffic was sent from a wired host to the mobile client as it navigates the route. Each packet was stamped with an ID, and the client recorded which packets were received successfully. The same test was repeated 20 times for the three roaming solutions. The traffic sent used a VoIP-like data profile of 240 byte packets sent at 33 packets per second. 4620 packets were sent during each circuit. Table 6 shows that Madwifi 0.9.1 fares very poorly. Madwifi 0.9.1-E performs significantly better, as expected. Madwifi 0.9.1-EFS is able to increase performance slightly in this case. The reason is that the full benefits of fast scanning are somewhat hidden in this test since the VoIP-like data does not push the limits of throughput.

	Mean Fraction Received	Median Fraction Received	Standard Deviation
Madwifi 0.9.1	0.41	0.41	0.16
Madwifi 0.9.1-E	0.70	0.69	0.05
Madwifi 0.9.1-EFS	0.77	0.76	0.04

Table 6: Multi-access point roam - UDP

3.7 Previous Work

Previous efforts have attempted to reduce scan completion time by reducing the number of channels scanned [**?**, **?**, **?**, **?**]. The idea is that each access point maintains a list of neighbors. When a station roams away from that access point, it only scans channels of neighbors thereby reducing scanning time. While this improves performance, it runs the risk of missing available access points. An alternative it is have the client do synchronized passive scanning [**?**] to learn about available basestations. While this reduces scan time, it adds some complexity to the infrastructure. Other research has optimized other aspects of roaming, such as authentication and IP address assignment, e.g. [**?**]. That work is orthogonal to the results presented in this paper.

3.8 Roaming Discussion

During the development of the fast roaming protocol, the wireless emulator testbed was used in a number of ways. First, the emulator allowed us to implement very simple experiments to quickly understand key properties of both the original and the improved versions of the driver. The GUI interface enabled an investigation into the source of the poor performance, by allowing us to execute interactive mobile experiments from the desktop. The ability to do debugging, testing, and performance tuning using (emulated) mobility while sitting at the desktop looking at the code resulted in a very fast development cycle. The alternative is to install code on a few of laptops and taking them around the building. This is certainly possible, but it is a slower and labor intensive process. Finally, we were able to compare the three drivers in identical but realistic scenarios, which would not be possible in the real world.

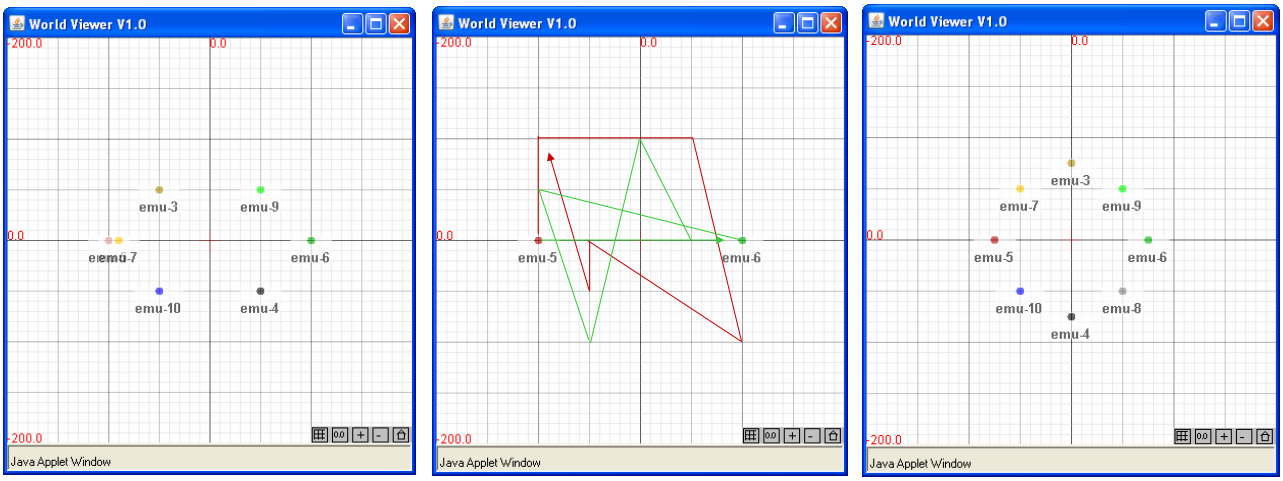
4. VIDEO STREAMING OVER 802.11

In our second case study, we evaluate the performance of time-aware MAC retransmission for video streaming.

4.1 Challenges

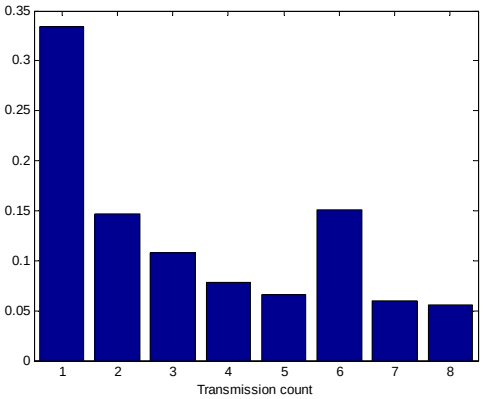
With the rapid adoption of 802.11 WLANs, wireless video streaming has gained a lot of attention. However, providing seamless wireless video streaming services is challenging. Poor channel conditions and interference can result in low bandwidth, high packet loss rates, and high delay, thus degrading the video quality. Moreover, the interframe coding used by many video coding standards means that even a single packet loss can significantly affect quality. We used the emulator to quantify the problem based on the topology shown in Figure 7(a). Nodes emu-3, emu-5, and emu-9 simultaneously run iperf UDP tests to emu-4, emu-6, and emu-10 for one minute. Madwifi 0.9.1 is used for all the tests. emu-7 is operating in monitor mode and is set up to have a perfect channel to emu-5, allowing it to monitor all

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3666
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3666

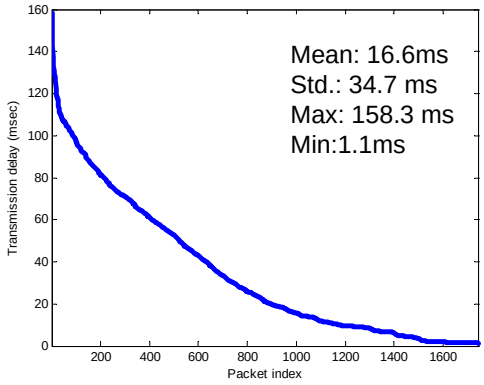


(a) General (b) Mobile-user (c) Congested-network

Figure 7: Streaming test topologies



(a) Transmission count



(b) Transmission delay

Figure 8: Transmission counts and delays

transmissions (including retransmissions) made by emu-5. All transmissions are timestamped using the 1 microsecond clock available on Atheros cards. Using emu-7's monitoring results, we can calculate the distribution of the transmission count and transmission delays for emu-5's transmissions

The results are shown in Figure 8. Transmission counts are the number of transmissions (initial attempts and retransmissions) issued from the sender for each packet and the transmission delay is the time needed, including transmit time, inter-frame space, and backoff interval. The results show that transmission delays fluctuate by more than two orders of magnitude. This is a result of the exponential backoff algorithm used by 802.11: once a packet has been retransmitted a few times, further retransmissions can take a very long time. The fact that the backoff counter is frozen while the medium is busy further exacerbates this effect when the traffic load is high. This high and variable packet delay is clearly a problem for real-time video streaming, since the receiver must display frames at a constant rate, so late frames must be discarded.

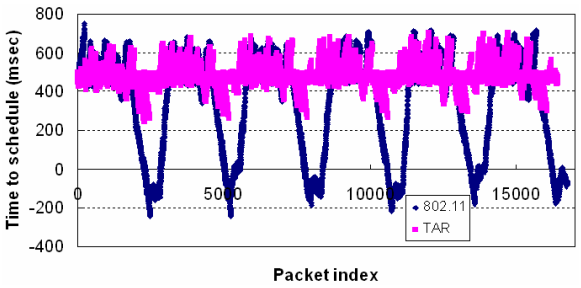
4.2 Time-based Adaptive Retry (TAR)

To reduce transmission delay, we have proposed TAR, a time-based adaptive retransmission mechanism [**?**]. In TAR, each packet is assigned a *retransmission deadline* at the application layer. This information is then used by the MAC layer to determine whether it should (re)transmit or discard the packet at some time point. This design philosophy is different from the conventional count-based strategy used in the 802.11 protocol in which a failed packet is retransmitted until a fixed retry limit is reached. For video, the retry deadlines are assigned based on the predictive-coding property of MPEG. Specifically, all packets in a group of pictures (GOP) are given the same deadline, corresponding to the end of the GOP. This means that reference video frames are assigned a relatively later deadline than inter-coded frames. This implicitly prioritizes important packets that contribute more to the video quality by giving them more transmission opportunities.

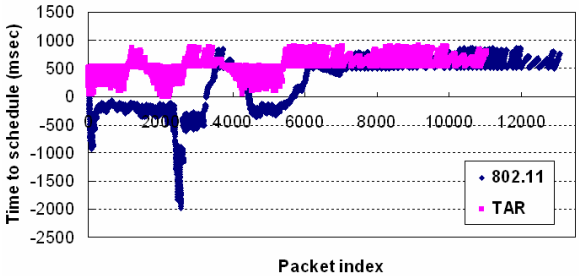
4.3 Evaluation

Earlier evaluations of TAR have used simulation. We now evaluate an implementation of TAR in the Madwifi driver using three scenarios. The first is a simple scenario with only the video server and client in the network. This sce-

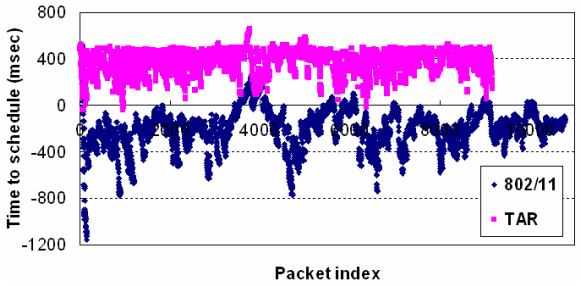
Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3666
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3666



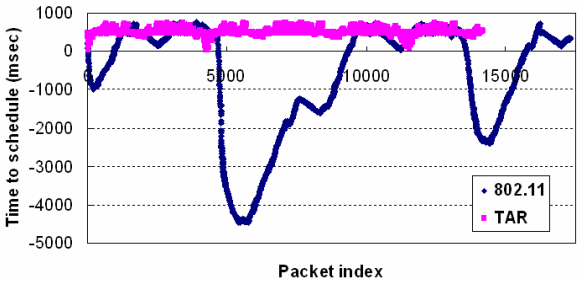
(a) Simple



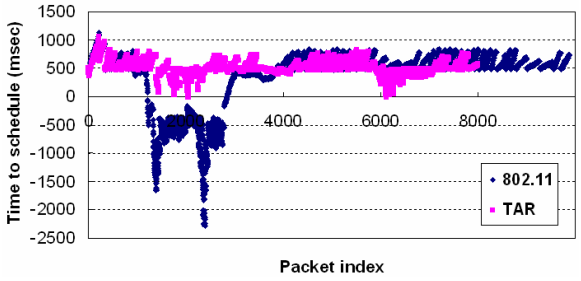
(b) Mobile-user



(c) Congested-network



(d) Mobile-user(simulation)



(e) Mobile-user(testbed)

Figure 9: Results of time relative to playback schedule

802.11	Eligible	Late	Lost
I frame packets	85%	9%	6%
P frame packets	78%	19%	3%
Total packets	79%	18%	3%
TAR	Eligible	Late	Lost
I frame packets	100%	0	0%
P frame packets	88%	0	12%
Total packets	90%	0	10%

(a) Simple

802.11	Eligible	Late	Lost
I frame packets	45%	32%	23%
P frame packets	50%	25%	25%
Total packets	50%	26%	24%
TAR	Eligible	Late	Lost
I frame packets	92%	0	8%
P frame packets	54%	0	46%
Total packets	59%	0	41%

(b) Mobile-user

802.11	Eligible	Late	Lost
I frame packets	7%	66%	27%
P frame packets	5%	56%	39%
Total packets	5%	57%	38%
TAR	Eligible	Late	Lost
I frame packets	99%	0	1%
P frame packets	46%	0	54%
Total packets	52%	0	48%

(c) Congested-network

802.11	Eligible	Late	Lost
I frame packets	54%	46%	1%
P frame packets	55%	45%	0%
Total packets	55%	45%	0%
TAR	Eligible	Late	Lost
I frame packets	97%	0	3%
P frame packets	80%	0	20%
Total packets	82%	0	18%

(d) Mobile-user (simulation)

802.11	Eligible	Late	Lost
I frame packets	47%	15%	38%
P frame packets	48%	9%	44%
Total packets	48%	9%	43%
TAR	Eligible	Late	Lost
I frame packets	95%	0	5%
P frame packets	50%	0	50%
Total packets	57%	0	43%

(e) Mobile-user (testbed)

Table 7: Results of Packet Ratios

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3666
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3666

	802.11	TAR
Simple	25.5 dB	32.0 dB
Mobile-user	22.1 dB	26.8 dB
Congested-network	7.8 dB	26.1 dB
Mobile-user(sim)	23.6 dB	30.7 dB
Mobile-user(testbed)	21.9 dB	26.5 dB

Table 8: PSNR values of the luminance component

nario was used mainly during debugging and testing of our TAR implementation. The second scenario is a mobile-user scenario (Figure 7(b)) with the video server and client moving at a velocity of 10 m/s based on a predefined route in a 100x100 square meter area. Ricean fading with $k = 3$ was used. The last scenario is a congested network (Figure 7(c)) in which six stations contend with the video server and client. Each contending station sends out a 1KB ping packet at a rate of 10Hz. In the tests, emu-5 and emu-6 serve as the video server and client, respectively. The video client drops late frames and displays the video according to its original coding rate without stretching or shrinking inter-frame intervals. The test sequence is "stefan" in CIF format, encoded with 30 frames per second and 15 frames per GOP (composed of an I frame and 14 P frames). The sequence is repeatedly streamed from emu-5 to emu-6 for one minute.

Figure 9(a)-(c) depicts the arrival time relative to the playback time for the three scenarios. The results show that TAR has nearly no late arrivals while 802.11 suffers a large number of late packets under degraded channel conditions. The increased percentage of lost packets in TAR reflects packets dropped by the MAC as a result of the time-based retransmission strategy. This translates into bandwidth savings by not transmitting useless information. Table 7(a)-(c) gives the ratio of eligible, late, and lost packets carrying different frame types. The results show that TAR prioritizes I frame packets over P frame packets through the assignment of retransmission deadlines. This unequal error protection translates into a higher PSNR for TAR ((Table 8): TAR outperforms 802.11 by more than 5 dB in PSNR for the video.

4.4 Related Work

Representative work on the MAC layer retransmission includes [**?**, **?**, **?**]. Li and van der Schaar [**?**] consider the MAC retry limit and sending buffer occupancy and suggest a heuristic for determining the operating point at which packet loss due to buffer overflow and link errors is minimal. Unlike [**?**], the goal of TAR is to solve the problem of late packets rather than transmit buffer overflow. Liebl et al. [**?**] develop an integrated scheduler and drop strategy in the link layer. The proposed method requires accurate channel state information, which is not easy to obtain in practice. [**?**] incorporates information about the video stream structure and future channel behavior into the scheduling algorithm. A solution based on a simplified rate-distortion model that relies on one single metric per user is proposed. TAR can be viewed as a special case of [**?**]. However, our approach does not rely on knowledge of future channel behavior or on the availability of rate-distortion-quality triples in the base station. All the above work uses simulation for evaluation.

4.5 Video Streaming Discussion

In [**?**], we demonstrated the effectiveness of TAR using simulation (OPNET). Simulation is a convenient tool to quickly evaluate a new protocol since the implementation is relatively easy. It is also possible to evaluate large topologies. However, simulation results may be misleading since they (over)simplify the wireless system. To explore this, we recreated the mobile-user scenario using OPNET modeler 12.0 [**?**]. OPNET allows us to define trajectories and add Ricean fading, as we did in the emulator. We also integrate our video server/client software into OPNET. The simulation results for the mobile-user scenario are shown in Figure 9(d) and Table 7(d). TAR again outperforms 802.11, but the simulation results show a greater performance difference between 802.11 and TAR. We discovered that this is the result of differences in how OPNET and the Linux driver implement flow control in the protocol stack. In Linux systems, the socket layer interacts with the user application to avoid buffer overflow and limit the queue size. However, OPNET's flow control results in large queue build up, which increases latency and hurts the performance of 802.11-based video streaming. TAR avoids the queue build up so TAR sees more substantial improvements in OPNET than on real hardware. This problem shows how simplifications in simulators can affect performance in hard-to-predict ways.

We also conducted the mobile-user experiment in the real world with two laptops using the same configuration as used in the emulator. Each laptop is carried by a person roughly following the trajectories in Figure 7(b) indoors. The channel is shared with other wireless stations so the test nodes are exposed to co-channel interferences. We repeated the experiment several times; Figure 9(e) and Table 7(e) show the results of one run. The high level result is similar to that of the emulator results, but the details are different, which is not surprising given the lack of control in this real world experiment.

During the implementation of TAR, the emulator greatly reduced the testing and debugging time. The interactive GUI can be used to explore different topologies quickly. We can also easily reproduce scenarios that trigger bugs or have poor performance. For example, we were able to quickly fix a memory leak in scenarios involving a gray zone by manually controlling path loss. This is hard to realize on an uncontrolled testbed. We were also able to perform extensive tests by varying the channel parameters at a fine-grained level. Finally, it is easy to set up a monitor node to listen in on the traffic of the experiment. By controlling the wireless channel, we can make sure the monitor node is not affected by interfering traffic.

5. CONCLUSION

In this paper we presented two case studies that used a testbed based on digital signal propagation emulation to support the implementation, testing, and evaluation of wireless and mobile systems. The use of a controlled testbed allowed us to gain some interesting insights. For example, we found that the roaming in the Madwifi driver was optimized for nomadic users and performed very poorly for mobile users. We also introduced a "fast roaming" solution that uses the wired network to collect probe responses efficiently. In our second study, we showed how time-aware MAC layer retransmission improved the quality of video streaming, es-

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3666
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3666

pecially for mobile users and congested networks.

Our experience showed that the wireless emulator was very useful during the various stages of the two projects. Early in the project, it allowed us to implement very simple tests to quantify key properties of the system. The interactive GUI also made it possible to quickly test the system in mobile scenarios while working at a desktop. Later in the project, the emulator was used to perform controlled and repeatable performance comparisons of different versions of the protocols under study. The video project also allowed us to compare emulation and simulation. While simulators offer a more pleasant development environment and support larger scale experiments, the level of realism they provide is hard to assess, especially with respect to timing at the lowest layers of the system.

6. REFERENCES

[1] C. Corporation. Cisco Compatible Extensions. `http://www.cisco.com/web/partners/pr46/pr147/partners_pgm_concept_home.html`.

[2] P. De, R. Krishnan, A. Raniwala, K. Tatavarthi, N. Syed, J. Modi, and T. Chiueh. MiNT-m: An Autonomous Mobile Wireless Experimentation Platform. In *Proc. of Mobisys 2006*, Uppsala, Sweden, June 2006.

[3] A. Dutta, S. Madhani, W. Chen, and O. A. andHenning Schulzrinne. GPS-assisted Fast-handoff for Real-time communication. In *2006 IEEE Sarnoff Symposium*, Princeton, March 2006. IEEE.

[4] E. Hernandez and S. Helal. RAMON: Rapid Mobility Network Emulator. In *Proc. of the 27th IEEE Conference on Local Computer Networks (LCN'02)*, Tampa, FL, November 2002. IEEE.

[5] G. Judd. Using physical layer emulation to understand and improve wireless networks, October 2006. PhD Thesis, Department of Computer Science, Carnegie Mellon University.

[6] G. Judd and P. Steenkiste. Using Emulation to Understand and Improve Wireless Networks and Applications. In *Proceedings of NSDI 2005*, Boston, MA, May 2005.

[7] G. Judd and P. Steenkiste. Software architecture for physical layer wireless network emulation. In *The First ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and CHaracterization (WiNTECH 2006)*, Los Angeles, Sep 2006. ACM.

[8] G. Judd and P. Steenkiste. Understanding Link-level 802.11 Behavior: Replacing Convention with Measurement. In *Wireless Internet Conference 2007 (Wicon07)*, Austin, Texas, October 2007.

[9] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions. In *Proceedings of MSWiM 2004*, Venice, Italy, October 2004.

[10] Q. Li and M. van der Schaar. Providing adaptive qos to layered video over wireless local area networks through real-time retry limit adaptation. *IEEE Trans. on Multimedia*, pages 278–290, April 2004.

[11] G. Liebl, H. Jenkac, T. Stockhammer, and C. Buchner. Joint Buffer Management and Scheduling for Wireless Video Streaming. In *Proceedings of ICN 2005*, April 2005.

[12] G. Liebl, K. Kalman, and B. Girod. Deadline-Aware Scheduling for Wireless Video Streaming. In *Proceedings of ICME 2005*, Amsterdam, the Netherlands, April 2005.

[13] M. Lu, P. Steenkiste, and T. Chen. A time-based adaptive retry strategy for video streaming in 802.11 wlans. *Wireless Communications and Mobile Computing, Special Issue on Video Communications for 4G Wireless Systems*, pages 187–203, January 2007.

[14] S. McCanne and S. Floyd. UCB/LBNL/VINT Network Simulator - ns (version 2). `http://www.isi.edu/nsnam/ns/`, April 1999.

[15] A. Mishra, M. Shin, and W. Arbaugh. An empirical analysis of the ieee 802.11 mac layer handoff process. *ACM SIGCOMM Computer Communication Review*, 33(2):31–31, 2003.

[16] A. Mishra, M. Shin, and W. Arbaugh. Context caching using neighbor graphs for fast handoffs in a wireless network. In *Proceedings of Infocom 2004*, Hong Kong, March 2004.

[17] B. Noble, M. Satyanarayanan, G. Nguyen, and R. Katz. Trace-based mobile network emulation. In *Proc. of SIGCOMM 1997*, Cannes, France, September 1997. ACM.

[18] OPNET Tech. Opnet. `http://www.opnet.com`.

[19] I. Ramani and S. Savage. SyncScan: Practical Fast Handoff for 802.11 Infrastructure Networks. In *Proceedings of Infocom 2005*, pages 675– 684, Miami, Florida, March 2005.

[20] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols. In *Proc. of WCNC 2005.*, New Orleans, LA, March 2005.

[21] M. Shin, A. Mishra, and W. Arbaugh. Improving the latency of 802.11 hand-offs using neighbor graphs. In *Proceedings of Mobisys 2004*, Boston, MA, June 2004.

[22] UCLA. Whynet Project. `http://chenyen.cs.ucla.edu/projects/whynet`.

[23] B. White, J. Lepreau, and S. Guruprasad. Lowering the barrier to wireless and mobile experimentation. In *Proc. of the First Workshop on Hot Topics in Networks*, Princeton, NJ, October 2002. ACM.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3666
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3666