Wired Wireless Broadcast Emulation

Daniel Havey, Roman Chertov, Kevin Almeroth
Department of Computer Science
University of California Santa Barbara
Santa Barbara, California 92316
Email: {dhavey, rchertov, almeroth}@cs.ucsb.edu

***Abstract*—A wired testbed's usefulness for wireless research hinges on its ability to faithfully reproduce the wireless medium. One of the key properties of a wireless medium is its broadcast nature. Wireless broadcast behavior is used in applications such as cell phone and satellite networks to disseminate the same data to multiple users as well as perform time synchronization. In this paper, we investigate two methods that can be used to ascertain if a given wireless emulator is modeling the broadcast property correctly. Our results indicate that the better of the two proposed methods offers micro-second precision.**

I. Introduction

Real-world protocols and applications often cannot be correctly evaluated using simulation. To solve this problem, researchers often rely on testbeds. Because testbeds are composed out of physical nodes, it is a simple matter of installing the desired protocol or application. However, in wireless research, testbeds face a number of serious challenges. Testbeds can be susceptible to environmental conditions and external, undesirable or non-reproducible interference. Wireless testbeds can suffer from limited topology options due to physical space constraints, for example, locating a number of wireless nodes in a small space can lead to unrealistic results. Finally, rearranging the testbed topology can be time consuming.

Wired testbeds are much easier to configure and are not prone to external interference issues. However, wired testbeds require an emulation component to be useful for wireless research. Naturally, the fidelity of the emulation is directly related to the accuracy of the results that the testbed can provide. Increased fidelity comes at a cost of increased configuration, processing, and complexity. This tradeoff occurs because emulators must be configured to accurately emulate the physical processes that are the result of communicating over a wireless medium. Accurately modeling the physical processes in real-time requires fast emulation nodes, and as a result, adds cost and complexity to the testbed.

Of critical importance is that not all wireless experiments require absolutely perfect emulation. Accurate results can be obtained with coarse approximations. One of the key principle differences between wired and wireless networks is the broadcast nature of the wireless medium. Wireless broadcast behavior is the underlying communication paradigm and also important for applications like cell phone and satellite networks because it is used to synchronize time and to disseminate control messages. The key property of a wireless broadcast is that a packet will arrive at all nodes within communication range at almost exactly the same time.

In this paper, we concentrate on exploring methods to measure broadcast propagation in a wired wireless emulation setting. Once it can be established that broadcast messages can be correctly emulated, it becomes possible to concentrate on other wireless properties such as interference, carrier sensing, and the impact of obstructions. However, if broadcast messages cannot be properly emulated, additional emulation complexity is irrelevant–the emulated behavior is already incorrect.

In a wired wireless emulator, broadcast messages might not arrive at the nodes at nearly the same time as nodes can be connected by a number of hubs and switches. This configuration can result in nodes receiving a broadcast message at varying times. Even if all the required testbed nodes are moved to a single switch and are assigned to the same VLAN, it is still possible for the nodes to receive broadcast messages at varying times. When a broadcast packet arrives at the switch, it has to be replicated and then forwarded to all of the ports in the VLAN. How quickly these operations can take place is a function of the switch's backplane processing capacity. Because of replication, it is entirely possible for some nodes in the VLAN group to receive the packet at different times. To measure this effect, we use *delay* as our metric. Delay in this context is the time required for a packet to leave the network driver at the source node, traverse the switch, and arrive at the network driver of the

receiving node. Comparing the delay values for multiple nodes receiving a single broadcast packet serves as an accurate indicator of how well the broadcast behavior was emulated in the wired testbed.

Measuring delay presents a significant challenge because of the problem of achieving highly accurate time synchronization. Terminals must all be closely time synchronized in order to ascertain if a broadcast packet has reached all of them at the same time. There are two primary resources available for micro-precision synchronization: the reference signal from a cell phone network and the Pulse Per Second (PPS) signal from the NAVSTAR Global Positioning System (GPS) satellite constellation. The reference signals can be accessed by the Network Time Protocol (NTP) daemon, which in turn skews the local clock towards the reference signal [7]. However, even with the best time synchronization, some amount of clock difference will still be present. We define *offset* as a measure of the clock differences between the testbed nodes.

In this paper, we present methods to measure *offset* and *delay* with micro-second precision. When developing these methods, we have made it a priority to use open-source software running on commodity hardware. This goal ensures a high degree of portability and applicability of the measurement methods that we developed. These methods can be used to establish the accuracy of a wired wireless emulator with respect to the accuracy with which wireless broadcasts are emulated. To evaluate the merits of our measurement methods, we created a wired wireless emulation testbed. Our preliminary results indicate the effectiveness of our measurement methods and the feasibility of accurately emulating broadcast behavior with micro-second accuracy.

The remainder of this paper is organized as follows. In Section II, we provide an overview of available wired wireless emulators in which our measurement techniques could be used. In Section III, we describe our measurement methods. In Section IV, we describe the components of our testbed and detail our time synchronization techniques. In Section V, we present our experimental results. Finally, we conclude in Section VI.

II. Related Work

Wired wireless emulators such as EMPOWER [9], and EMWIN [8] use a centralized emulation layer. Even though these centralized approaches can avoid time synchronization issues because everything happens on a centralized controller, these approaches rely on switching equipment to disseminate messages to end-user nodes. Variability in the centralized controller or the switching equipment can result in broadcast messages being received at varying times.

Non-centralized wireless testbed emulators such as SEAWIND [6], SWOON [4], and the Scenario Driven Wireless Emulator by Bateman et al. [1] rely on a wired configurable testbed similar to University of Utah's Emulab[1]. These testbed emulators rely on various link shaping techniques to approximate a wireless link. Typically, a special node is used for one or more links that need to be emulated. Since these testbeds utilize switching equipment and multiple nodes to propagate wireless broadcast messages, the quality of the wireless emulation can suffer if the broadcasts are received at widely different times.

These different approaches to wired wireless emulation all have their various advantages and disadvantages. However, in each case, researchers using these emulators could benefit from our measurement techniques to validate their approaches. This is particularly critical when performing experiments using protocols that rely on the broadcast nature of the wireless medium such as Code Division Multiple Access (CDMA) when it is used in cellular networks.

III. Methodology

We present two methods to measure the precision of the time synchronization necessary to compute offset values. Acquiring the relative clock offsets from the testbed nodes is necessary to establish how closely they are synchronized. The better the synchronization, the more accurate the delay measurements become as they are computed by taking the time difference between when nodes send and receive a packet.

A. Direct Offset Measurement

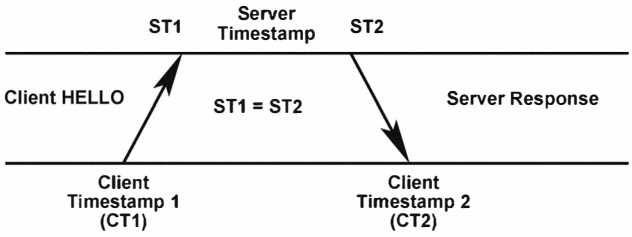


Fig. 1. Offset measurement timing diagram.

Figure 1 demonstrates the process of computing clock offsets between two separate nodes. A client program

[1]http://www.emulab.net/

records a timestamp at *CT1* and then sends a HELLO packet to the server. When the server receives the packet, it records a timestamp at *ST1*. It then takes another timestamp at *ST2* and returns its response to the client. When the client receives the server response, it records a final timestamp at *CT2*. The algorithm calls for the server to record a second timestamp at *ST2*. However, in practice, we find that consecutive timestamps yield the same time, so *ST2* = *ST1*, and we only return a single timestamp to the client. This method is based on the NTP algorithm [7], [2].

In the equation below, *(CT1 - ST1)* represents the time required for a packet to travel from the client to the server. The term *(CT2 - ST2)* is the time required for the packet to travel from the server to the client. The term *CT1* occurs before *ST1* and *CT2* occurs after, so these terms are of opposite sign leaving only the *Offset* if the network is symmetric.

$$Offset = \frac{(CT1 - ST1) + (CT2 - ST2)}{2} \quad (1)$$

B. Composite Delay and Offset Measurements

In the diagram of Figure 2, we see four nodes connected via a switch. *Node 1* broadcasts packets to *Nodes 2-4*. *Ts1* is a timestamp and is logged when a packet is broadcast from *Node 1*. *Tr2-4* are timestamps and are logged when a broadcast packet is received at *Nodes 2-4*.

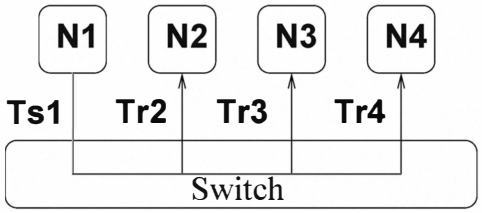


Fig. 2. Composite delay and offset measurements.

In the following discussion, we define timestamp values as $T_{event}Node$, where $event$ signifies an event when the timestamp was taken, and $Node$ identifies the testbed node at which the timestamp was taken.

In the equation below, we observe that the offset between a sending node and a receiving node, O_{S_R}, is equal to the time when the packet was sent at the sending node $T_{sent}S$ minus the time at the receiving node when the packet was sent$T_{sent}R$.

$$O_{S\_R} = T_{sent}S - T_{sent}R \quad (2)$$

The difference between a timestamp $T_{recv}R$ logged when a packet reaches the receiving node and a timestamp $T_{sent}R$ logged when the packet was transmitted from the sending node is composed of delay and offset values. The delay, D_{S_R}, is the time required for a packet to traverse the switch from the sending node to the receiving node. The offset is equal to O_{S_R} as described in Equation (2).

$$T_{recv}R - T_{sent}R = D_{S\_R} + O_{S\_R} \quad (3)$$

Even though this method computes the time difference, which is composed of delay and offset values, *it is critical to note that if either value can be treated as a constant, then the other value can be easily computed.*

IV. Experimental Setup

In this section, we provide a detailed description of the components of our testbed as well as an overview of the GPS clocks that we used.

A. GPS Time Synchronization

To minimize the clock offset in our testbed, we used open source software connecting commodity GPS receivers to an RS-232 serial interface on each of the terminals.

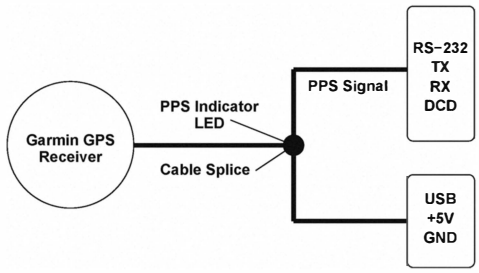


Fig. 3. GPS to RS-232/USB interface diagram.

Figure 3 shows a Garmin LVC 18 bare wire GPS receiver that is connected to an RS-232 interface for data acquisition and a USB port for power. We had to manually splice the cables as this GPS receiver is primarily designed for marine navigation and not Pulse Per Second (PPS) signal acquisition. The PPS signal is carried through the Data Carrier Detect (DCD) pin on the RS-232 interface along with the transmit and receive signals in accordance with the RS-232 International Telecommunication Union (ITU-T) specification.

The GPS receiver communicates with an NTP daemon by sending National Marine Electronics Institute 0183 (NMEA) sentences over the RS-232 interface. The open source SHared Memory PPS (SHMPPS) driver receives

the PPS signal from the GPS on the RS-232 DCD interrupt. SHMPPS calculates reference clock information and sends it to NTP. The NTP daemon gradually skews the local PC clock towards the SHMPPS clock reference. The local PC clock has a tendency to oscillate, thus requiring NTP to keep it in check. Even if two or more nodes are GPS clock synchronized, they can have varying offsets between each other because their local clocks can oscillate differently.

B. Testbed Hardware

In Figure 4, four time servers *(TS1-4)* are shown. In our testbed, each was constructed from a Dell Precision 390 with dual core Core2 Duo Intel processor. Each time server was GPS synchronized as described above and has two network interfaces, one was built into the motherboard and was connected to the control plane (not shown). The other, an Intel Pro 1 Gbps Ethernet card was connected to the experimental plane.

In addition to the time servers, we constructed two traffic generator nodes *TG1-2* from Dell Power Edge SC 1430s, each with two quad core AMD Opteron 2350 processors. As with the time servers, the onboard gigabit Ethernet interface was connected to the control plane switch. *TG1* had two quad port Intel Pro 1 Gbps Ethernet cards connected to the experimental plane and *TG2* had one. All of the testbed nodes use the Click Modular Router [5] software for packet generation and capture.

The experimental and the control plane were constructed from identical Cisco Catalyst 3560 series VLAN capable gigabit switches.

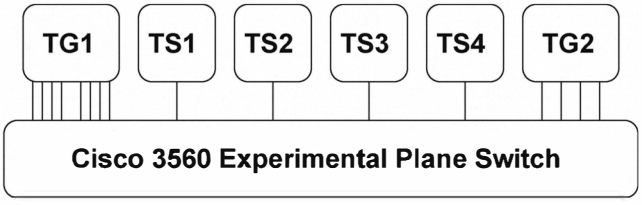


Fig. 4. Testbed configuration.

The network device drivers were configured for packet timestamping as described in the work by Chertov et al. [3]. This configuration ensures that packets were timestamped as early as possible to minimize the effect of processing delays on the timestamp.

V. Experimental Results

This section presents the results from a series of experiments designed to determine if the direct offset or the composite delay and offset method provides the better results. Using the better method, we then conducted a series of measurements to determine how closely in time the testbed nodes received broadcast packets.

A. Direct Offset Experiments

We connected RS-232 cables from *TS1* to *TS2*, *TS3*, and *TS4*. We chose RS-232 cables as they provided a direct link between the machines and hence there was no cross traffic to impact the measurements. *TS2*, *TS3*, and *TS4* nodes performed message exchange with *TS1* as described in Section III-A.

Figure 5 presents offset values computed via Equation (1) between *TS2* and *TS1* over a 10 minute period with samples taken once per second. The x-axis shows time in seconds and the y-axis shows offset in microseconds. A zero value indicates identical times, non-zero values imply that one clock is slower or faster than the other one. Ideally, offset should always be zero.

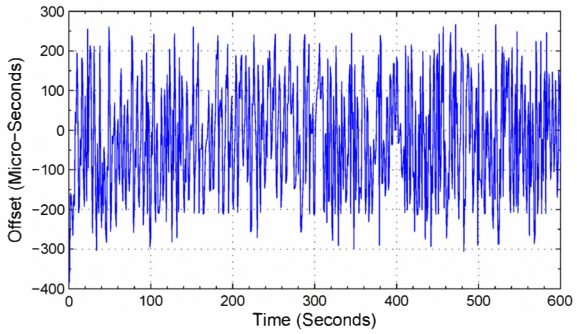


Fig. 5. Clock offset between *TS2* and *TS1*.

Figure 5 shows the offset values oscillating between -300 μs and 250 μs. Similar behavior was observed for other nodes as well. We believe that this large amount of variability is due to either to the RS-232 drivers or the cards themselves. However, the experiments showed that our GPS driven time servers were synchronized with sub-millisecond precision.

B. Composite Delay and Offset Experiments

a) Delay Measurement: The composite measurement approach described in Section III-B can be used to measure delay or offset if one of the values is known. In order to eliminate offset and measure delay, we used *TG1* (see Figure 4) to broadcast a CBR UDP flow from one interface. Since the packets had a broadcast MAC address, the switch replicated them and sent a copy to each of the other seven interfaces on *TG1*.

The experiments were performed with and without cross traffic. The experiments lasted for three minutes,

TABLE I
DELAY MEASUREMENTS WITHOUT CROSS TRAFFIC.

Rate	*eth1*			*eth2*			*eth3*			*eth4*			*eth5*		
	10^{th}	50^{th}	90^{th}	10^{th}	50^{th}	90^{th}	10^{th}	50^{th}	90^{th}	10^{th}	50^{th}	90^{th}	10^{th}	50^{th}	90^{th}
1	14	15	18	17	19	22	14	14	15	18	20	23	13	14	15
10	14	14	18	15	18	22	14	14	15	18	19	22	13	14	14
100	14	15	19	14	18	20	14	14	15	18	19	21	13	14	15
500	14	15	19	14	18	20	14	14	15	18	19	21	13	14	14

TABLE II
DELAY MEASUREMENT WITH CROSS TRAFFIC.

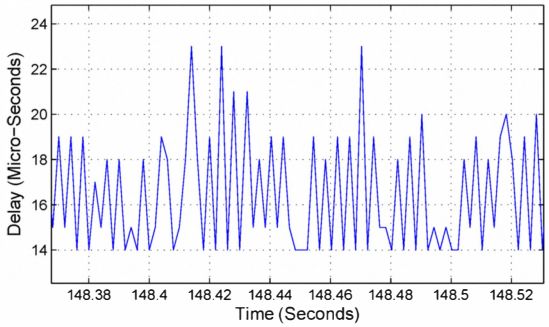
Rate	*eth1*			*eth2*			*eth3*			*eth4*			*eth5*		
	10^{th}	50^{th}	90^{th}	10^{th}	50^{th}	90^{th}	10^{th}	50^{th}	90^{th}	10^{th}	50^{th}	90^{th}	10^{th}	50^{th}	90^{th}
1	15	16	22	16	19	22	15	15	16	18	20	22	14	15	16
10	14	15	21	15	18	21	14	14	15	17	19	21	13	14	15
100	14	15	20	14	18	21	14	14	15	17	19	21	13	14	14
500	14	15	20	14	18	21	14	14	15	17	19	21	13	14	14

and we repeated every experiment five times. Finally, we used 64-byte sized packets and used the following packet rates: 1, 10, 100, and 500 packets per second. Cross traffic was generated by sending CBR UDP flows from the four interfaces in *TG2* to *TS1-4* nodes. The four flows send 64, 256, 1200, and 1500-byte packets respectively at 500 packets per second. The delay values were calculated by using the Equation 3 *with the offset set to zero as the broadcast packets originate and terminate on TG1*.

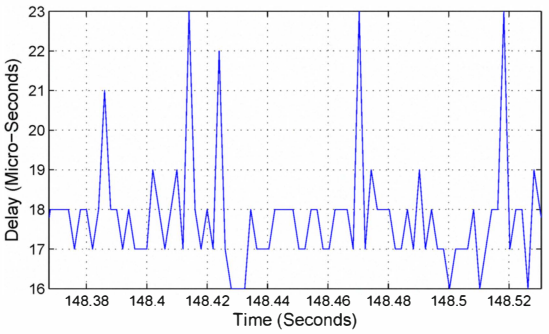
Figure 6 shows *TG1*'s raw delay data over time for two interfaces when cross traffic was not used. The data shows little variance with values ranging from 14 μs to 23 μs.

If the variance in delay is small, we can treat it as a constant and use it to calculate offset values using Equation 3. Table I and Table II present the $10^{th}, 50^{th}$, and 90^{th} delay percentiles with and without cross traffic, respectively. The tables only present the values for five of the seven interfaces. The other two were not included for the sake of brevity; however, their values are similar. From these results, we have made two observations. First, the difference between the 10^{th} and 90^{th} percentiles is very small, implying low variance. Second, the delay values are very similar with and without cross traffic. Although, there were delay values recorded as high as 4167 μs in the experiments without cross traffic, and 3625 μs in the experiments with cross traffic, these values occurred extremely rarely and were most likely caused by the operating system or hardware delays. The data indicates that our Cisco 3650 experimental switch performs broadcast replication very well and we can treat delay as a constant. From the data, we compute delay to have an 18 μs value.

The previous experiment has characterized delay. It is nearly constant indicating that a wired wireless testbed



(a) Interface *eth2*



(b) Interface *eth6*

Fig. 6. Broadcast delay measurements w/o cross traffic.

can emulate wireless broadcast behavior with a high degree of accuracy. This result allows us to proceed with our next experiment to measure the residual clock offset.

b) Offset Measurement: Although the experiment shown in Figure 5 demonstrated the sub-millisecond precision of our time servers it did little to show *Offset* behavior at sub-millisecond time scales. To investigate these effects, we broadcast UDP packets in a CBR flow from *TS1* to *TS2-4*. Using Equation 3 and the 18 μs delay value calculated previously, we calculate offset value for each node pair. Just as before, the experiments lasted three minutes and were repeated five times. *TS1* was

configured to send a broadcast flow of 64-byte packets with the following packet rates: 1, 10, 100, and 500 packets per second. Cross traffic was generated by *TG2* using the same parameters described previously, except that the sink was changed from *TS1-4* to *TG1*.

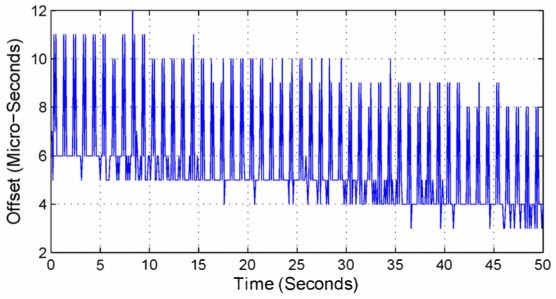


Fig. 7. Offset values at 10 packets per second with no cross traffic.

Figure 7 presents the raw data results from an offset experiment without cross traffic, and Figure 8 presents the results from the experiment with cross traffic. Both graphs are computed from the *TS2* node data. The x-axis describes the experiment time in seconds and the y-axis is for offset values as measured in microseconds. Both graphs show offset values well below the $+/-300$ μs values from the experiment in Section V-A. The slight jitter is caused by the variance in delay observed in the previous experiment and the slight drift of the clocks.

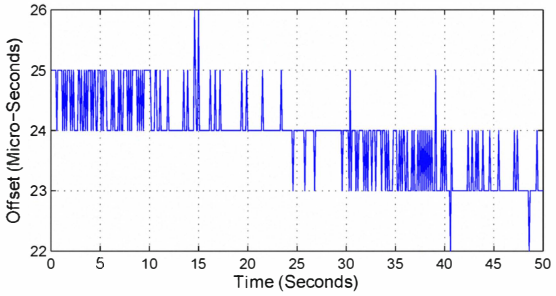


Fig. 8. Offsets at 10 packets per second with cross traffic.

Over all of the experimental runs, the 10^{th}, 50^{th}, and 90^{th} offset percentiles are 7 μs, 18 μs, and 53 μs, respectively. The maximum offset that we observed was 667 μs; however, values larger than 53 μs occurred extremely rarely. The data indicates that the nodes are synchronized to less than 100 μs of each other, which is a much better result than obtained in Section V-A for the first method.

The delay and the offset measurement experiments have shown that a wired wireless testbed can emulate wireless broadcast behavior with a very high degree of accuracy, and that it is possible to achieve very close time synchronization by relying on GPS reference signals.

VI. Conclusion

In this paper, our goal was to determine if wireless broadcast behavior can be accurately emulated in a wired environment. Specifically, we were interested in determining if in a wired testbed, a broadcast message will be received at nearly the same time by multiple nodes. We were interested in identifying validation methods that could be used by other wireless researchers to determine the accuracy of broadcast emulation in their testbed. For this purpose, we constructed a simple GPS-synchronized testbed and carried out validation experiments. Our results indicate that it is possible to emulate broadcast behavior with a high degree of accuracy.

In future work, we plan to investigate the accuracy with which a wired wireless testbed can emulate the collision avoidance behavior of wireless networks. Understanding collision avoidance behavior emulation along with the insights gained in this work will provide a foundation for building a high-fidelity wired wireless emulator.

References

[1] M. Bateman, C. Allison, and A. Ruddle. A scenario driven emulator for wireless, fixed and ad hoc networks. In *Proc. 4th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet)*, June 2003.

[2] J. Burbank, W. Kasch, J. Martin, and D. Mills. Network time protocol version 4 protocol and algorithms specification. Internet-Draft http://www.ietf.org/internet-drafts/draft-ietf-ntp-ntpv4-proto-11.txt, March 2009.

[3] R. Chertov, S. Fahmy, and N. Shroff. A device-independent router model. In *Proc. of IEEE INFOCOM*, April 2008.

[4] Y. Huang et al. Swoon: a testbed for secure wireless overlay networks. In *Proc. of USENIX Cyber Security Experimentation and Test (CSET) Workshop*, pages 1–6, July 2008.

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, Aug 2000.

[6] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko, and K. Raatikainen. Seawind: a wireless network emulator. In *Proc. of 11th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, September 2001.

[7] D. Mills. Internet time synchronization: The network time protocol. *Communications*, 39(10):1482–1493, Octeber 1991.

[8] P. Zheng and L. Ni. Emwin: emulating a mobile wireless network using a wired network. In *Proc. of ACM Workshop on Wireless Mobile Multimedia (WOWMOM)*, pages 64–71, September 2002.

[9] P. Zheng and L. Ni. EMPOWER: A network emulator for wireline and wireless networks. In *Proc. of IEEE INFOCOM*, volume 3, pages 1933–1942, March 2003.