Experimentation-Oriented Platform for Development and Evaluation of MANET Cross-Layer Protocols

Guevara Noubir, Wei Qian, Bishal Thapa, Yin Wang

College of Computer and Information Science
Northeastern University
Boston, MA 02115.

Abstract

[1] We introduce a platform that simplifies the development, performance evaluation, and comparison (DEC) of cross-layer protocols for multi-hop ad hoc networks (MANET). To the best of our knowledge, this is the first platform that provides an integrated API for controlling and assessing, on a per-packet basis, the physical layer and MAC/LLC parameters (e.g., frequency channel, power, rate, coding/modulation, fragmentation, number of retransmissions) for IEEE802.11 network interfaces.

Today, performance evaluation of wireless mobile ad hoc networks heavily relies on simulation, which is limited in reflecting real environment scenarios due to the channel modeling approximations. Experimental comparisons of protocols in real world propagation environment are difficult because of limited reproducibility of the channel propagation, environment, and mobility scenarios. We introduce a DEC platform that provides a network virtualization above the physical/link layers. It runs multiple protocols in TDMA-like timeslots guaranteeing an equitable share of the medium, seamless switching between protocols and synchronization between the nodes, all in a transparent fashion to developers. An extensive experimental evaluation demonstrates the usefulness of the platform.

Key words: MANET, Cross-Layer Design, Evaluation Platform, Testbed, IEEE802.11, Protocol Performance

[1] This research was supported by NSF CAREER AWARD CNS-0448330.
Email address: `noubir, qwlli, bthapa, yin @ccs.neu.edu` (Guevara Noubir, Wei Qian, Bishal Thapa, Yin Wang).

1 Introduction

During the last decade, research in multi-hop wireless ad hoc networks (MANET) has become one of the most active areas in networking research. Various protocols (i.e., DSR, AODV, ZRP, OLSR, TORA [9]) have been proposed and many techniques were investigated to address issues such as scalability, robustness, security, and energy conservation. Recently, cross-layer design has become a popular approach to optimize the resource usage in MANET. Such techniques make use of a coordinated adaptivity of the physical layer, MAC layer, and routing schemes. A cross-layer protocol needs to decide on a per-packet basis what frequency channel, power level, rate (i.e., coding and modulation), fragments size, and number of MAC retransmissions are to be used. They also require an accurate estimation of the channel and medium status. Although significant research has been done on the design and simulation of such protocols for MANETs [6, 12, 15, 16, 23, 44, 49], there were no tools that allowed the development of a cross-layer protocol which controls parameters such as coding scheme, modulation, frequency, and transmission power level on a per packet basis. Our platform provides such services to the protocol developer.

On the performance front, the current evaluations of MANET protocols mainly relies on simulation, but little on experimental comparison. Modeling, analysis, simulation and real-world experimentation are the four necessary steps for developing and evaluating a protocol. In this process, the experimental comparison is as important and necessary as simulation. This is because realistic modeling of wireless networks is intrinsically difficult. Several factors such as the geography, interference, hardware sensitivity, and mobility can have a dramatic impact on the link quality [1]. The experimental evaluation of MANET has been quite limited because it suffers from difficulty of reproducing the propagation environment due to rapid changes in channel conditions. We did several experiments to verify this. For example, we set the source to periodically broadcast (no retransmission) packets with a sequence number while the receiver collects them and records the loss rate and Received Signal Strength Indication (RSSI) for each received packet. After 10 minutes, without changing their positions, we repeat the same experiment again. We find substantial differences in RSSI and Packet Loss Rate between two experiments run consecutively in both indoor and outdoor cases [See Figure 1]. Thus, even a few minutes gap can become a huge obstacle in reproducing the propagation environment. For details of this experimental setup, please refer to Section 3.

In this paper, we propose a platform that aims at comparing multiple protocols in an "almost identical" propagation environment. It provides a physical/link layer virtualization that allows to run multiple protocol stacks in an interleaved manner. It enforces fairness between the various protocol stacks by scheduling them in a TDMA-like fashion, which also has the advantage of preventing a stack from interfering with another stack globally. So far, we have focussed on the virtualization and comparison of routing and lower-layer protocols. However, our platform can be extended to support stacks covering the transport layer. The platform is responsible for the node synchronization and time virtualization, which allows a transparent implementation of protocols. With this approach, the protocols experience the similar changes in macro-characteristics of the channel propagation. For example, the changes in frame loss-rate and RSSI that happens within a second is much lower compared to the changes during a 10 minutes span [See Figure 1]. Since the interleaving period we choose for running experiments on the platform are within a second (250ms-500ms), the protocols undergo similar non-instantaneous channel propagation effects.

The main assumption of our platform is that the coherence time for the channel macro-characteristics (loss rate, RSSI) is in the order of the interleaving period. We believe that this is a reasonable assumption because in a mobile environment, obstructions (e.g., a car passing by, a door closing) and, in general, the geography and interference do not appear and disappear instantaneously. Clearly micro-characteristics of the channel are not stable, but they do not impact the average link quality over the scheduling period. We did an extensive experimental evaluation of our platform by comparing two instances of a MANET protocol. Highly correlated performance between instances in the interleaved case against weakly correlated performance in the non-interleaved case confirms this assumption.

There are several drawbacks to the experiment-based protocol evaluation methods which run different protocols separately and compare their performance. For example, reproducing the exact mobility patterns and channel conditions, avoiding hardware failures and battery drain, and running a large number of experiments to reach a statistically stable result. Our platform aims at resolving those issues.

Contributions:

- We developed a platform for simplifying the development of cross-layer MANET protocols. The platform allows to control the physical layer and MAC layer parameters on a per-packet basis (e.g., transmission power level, frequency channel, coding/modulation). It also allows a fine grain channel and medium status assessment.
- It provides an “almost identical” propagation environment with the transparency in implementing protocol stacks (without knowledge of virtualization and time scheduling details) for comparing multiple protocols. It also features node synchronization, resource usage monitoring, and protocol stacks deployment.

We introduced the idea of our platform in [43]. Now, the complete system is available to the wireless community along with the experimentation data from the evaluation [51].

The paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we motivate our work by showing the channel irreproducibility in the case of single-hop communication. In Section 4, we describe the architecture of our framework and its main components. In Section 5, we describe the techniques we used to evaluate our framework, and present the results of the evaluation.

2 Related Work

Cross-layer Design: Cross-layer protocol design has gained substantial interest in the last few years. Various protocols that require a coordinated effort from the network stack have been proposed. However, most of the protocols that require an access to PHY/MAC layer control and assessment were limited to the simulation [6,12,15,16,23,30,44,49]. Implemented cross-layer protocols were limited by the inability to adequately control the physical layer parameters on a per-packet basis. From the simulation results it appeared that significant energy saving and capacity increase could be achieved if tools supporting the implementation of cross-layer optimization were available [2,4,17]. This motivated us to develop a platform that provides the tools for cross layer-protocol development and evaluation.

Protocol Evaluation: On the evaluation aspect of protocols, the performance evaluation of MANET protocols has mainly relied on simulation [39, 41,45,55]. Simulation has the advantages of reproducibility, lower-cost, time-efficient evaluation and is more feasible for larger network deployments. However, simulation environments have many limitations:

- *Several flaws* have been discovered. For example, the most widely used mobility model for simulation, called the random waypoint model, was shown suffering from a speed decay problem [8, 53]. Other problems were also pointed out in [13]. Although several solutions have been proposed [10, 32, 54], it is still very difficult to model and simulate real-world mobility patterns.
- Existing simulation environments are *not always consistent*. In [10], the performance of a flooding protocol was shown to significantly diverge from one simulator to another (i.e., OPNET, GloMoSim, NS-2).
- The physical layer *modeling is limited*. Many simulation environments do not accurately model the physical layer propagation (e.g., terrain influence, fading, shadowing) [50]. The interference is inaccurately taken into

account in NS-2 to compute the Bit Error Rate (BER) and Frame Error Rate (FER) [11, 15, 29]. This translates into performance that is not always realistic compared to the real-world performance [27].

- In the case of IEEE802.11 simulation, the *physical layer and link layer are not always fully implemented*. Some simulation environments do not implement all transmission rates. Simulators that support multiple transmission rates impose that only a single rate can be used throughout the simulation period.
- Finally, different transmission rates should correspond to different receiver sensitivity thresholds (as specified by most IEEE802.11 cards manufacturers). However, current simulation environments only allow to specify a single sensitivity threshold for all rates. For a list of reasons that explains the discrepancies between simulation and field measurements one can refer to [3, 11, 26, 31].

An alternative to real-world experiments could be using channel emulators to connect the RF front ends. However, such instruments are expensive. Therefore, such approach is not appropriate when a large number of links is needed as is the case in MANETs. Now, the experimental comparison of MANET protocols also has an issue [1, 11, 15, 29, 35, 46]. It suffers from the difficulty of reproducing the same channel conditions for multiple protocols under evaluation. An interesting approach taken in [19, 22, 42, 47] aimed at developing hardware/software platforms that allow to carry experiments in a remote controlled reproducible environment. This allows more accurate performance comparison of protocols. Our platform is a tool for evaluation of protocols in a real-world setting that any research team with a set of laptops can use. Our platform modifies and builds on the highly popular Madwifi driver in a similar way as SoftMAC [52] but for a different purpose.

3 Unpredictability of Single-Hop Communication

To motivate the usefulness of our approach, we first ran a few sets of experiments to characterize single-hop communication. We tried both static-indoor and mobile-outdoor scenarios. In both scenarios, the nodes operate in the ad hoc mode. The source periodically (every 50ms) broadcasts packets with a sequence number. The receiver collects the received data and determines the frame loss rate per second [See Figure 1]. The transmission power level used for the indoor and outdoor experiments is 5mW (7dBm) and 50mW (17dBm) respectively. The collected data was part of a part of a larger experiment to characterize the IEEE802.11 wifi interfaces for multiple power-levels and rates [51].

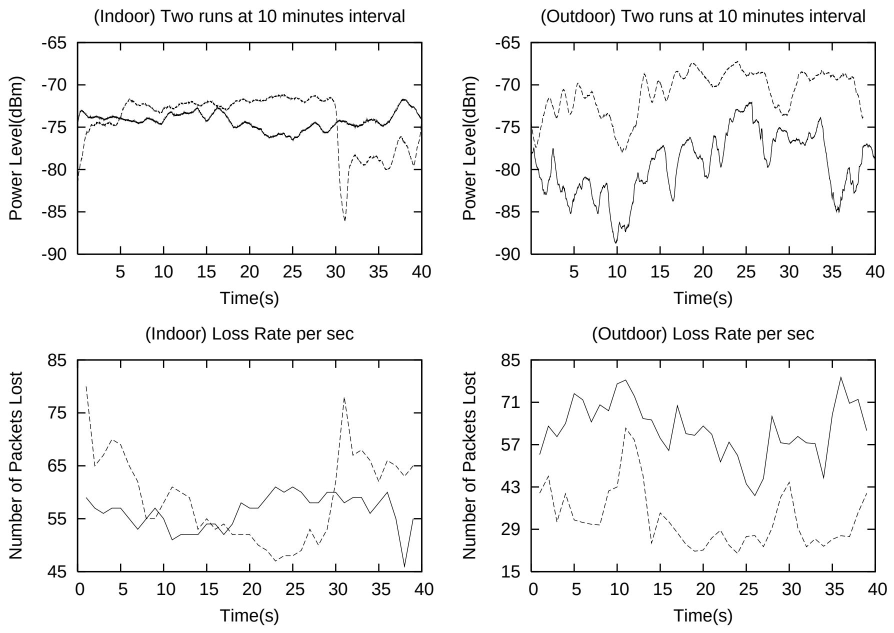


Fig. 1. RSSI and Packet Loss Rate comparison between two experiments at 10 minutes interval with 2.5K packets/sec. (Left::Indoor) Computer Science building, Northeastern University. (Right::Outdoor) Museum of Fine Arts, Boston.

The static-indoor experiments were performed inside the WVH building in Northeastern University. We observe that the RSSI [2] and the number of packets lost within a second were mostly stable, however, there were huge fluctuations inside a 10 minutes interval [See Figure 1]. Several explanations could be given, such as microwave oven activity or neighboring access points load (i.e., co-channel and adjacent-channel interference) or human activity (e.g., doors closing, people moving) resulting in multi-path fading.

The mobile-outdoor experiments were run around the Museum of Fine Arts in Boston. The mobility covered an area of approximately (50 meters $\times$ 300 meters). While running each of the outdoor experiments, we made every possible effort to follow exactly the same mobility pattern with minimal human perturbation. The experimental results indicate that the loss is very sensitive to the environment conditions such as, vehicular traffic (trains, buses, cars), people moving around, and the physical orientation of the device.

We conclude that it is very difficult to predict and reproduce the physical layer characteristics over a long period of time. Therefore, comparing two protocols by running them separately over a 5 minutes span, while trying to reproduce the mobility pattern as accurately as possible, still leads to unreliable and unpredictable results. This observation is further verified in section 5.

[2] According to the Atheros chipset description the received power in dBm is $P(dBm) = RSSI - 95$, where RSSI is provided by Atheros firmware for each received packet.

4 Virtualization Architecture and Components

Our proposed framework allows the comparison of multiple protocol stacks in a transparent way to the protocol designer and implementer. Our focus has so far been routing and lower-layer protocols, however the platform can be extended to higher layers. It also provides several tools for controlling physical layer parameters, protocols deployment, and experimental data logging/gathering. The framework does not depend on a particular physical layer implementation. As long as the physical layer API supports the dynamic selection of frequency channel, power level, rate (i.e., combination of modulation and coding), it can be straightforwardly supported by the framework. The framework has been implemented as a platform that runs on a testbed of Linux laptops. It uses Madwifi [33] driver and supports the IEEE802.11a/b/g chipset from Atheros. The Madwifi driver is compatible with cards produced by over 50 manufacturers [34]. The platform works within the click modular router environment [14, 28]. We first outline the platform architecture and its components, then we discuss them in greater details.

Virtual environment: Each instance of a *physical protocol* is called a *virtual protocol.* A virtual protocol has access to a *virtual link* layer, and to the *virtual time.* The communication between the virtual protocol stack and the virtual link is accomplished through two queues (In/Out). The packets are multiplexed/demultiplexed by the platform. The *protocol scheduler* interleaves multiple instances of a set of protocols specified by the user. The protocol scheduler guarantees that at each instant of time, same virtual protocol is running across the whole network. All reference to time (e.g., timers) is intercepted by the platform and virtualized. The running instance has only access to the virtual time. Therefore the underlying interleaving has no impact on the protocol mechanisms such as timeouts and retransmissions. The *Virtual link scheduler* is synchronized with the protocol scheduler. It guarantees that during each timeslot only the packets of the *active* protocol are considered by the MAC/Physical layer. Finally, new protocols can be straightforwardly implemented and integrated within the platform.

MAC/Physical layer control: The platform also provides an enhanced API to the physical layer (driver/firmware) to specify for each packet the desired transmission frequency channel, power-level, rate (i.e., combined modulation/coding scheme), fragmentation threshold, and maximum number of retransmissions to be used.

Peripheral tools: In order to operate the platform correctly, the nodes are synchronized using a driver-level light-weight protocol that achieves a synchronization below $15\mu s$. This level of synchronization allows us to support interleaving period of few hundreds of milliseconds. The driver also incorporates a monitoring service that gathers information about all the packets transmitted (for e.g., number of retransmissions, rate, power-level, timing, RSSI of received packets).

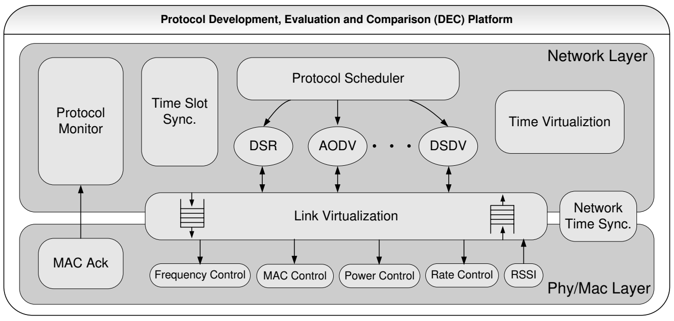


Fig. 2. Platform Architecture.

4.1 Experimentation Testbed: Hardware and Software

The testbed used for the experiments consists of several IBM laptops. They are equipped with Madwifi compatible IEEE802.11 wireless cards (Netgear WAB501 in this case). Each node runs an instance of the platform built within the Click modular router environment.

Operating System: The experimentation laptops used Red Hat Linux with the kernel version 2.4.24. The default tic-time [3] of the kernel is 10 milliseconds. We modified this value to 1 milliseconds in order to achieve an adequate-accuracy for the protocol scheduling in our platform.

Wireless Interface: The platform supports dynamic frequency, power, rate, modulation and coding control on a per packet basis. Traditional approaches to dynamic frequency, power and rate control rely on `iwconfig` command [24]. But its usage results in resetting the wireless adapter, which in turn disrupts the normal network traffic [17]. This results in a broken link and can take more than one second to recover. This approach would make per-packet control impractical for a cross-layer protocol. In our platform, we take advantage of the available and modifiable features of Madwifi to resolve this issue.

Madwifi is an open source Linux kernel driver for wireless LAN devices. It is compatible with a large number of cards (over 100) manufactured by over 50 companies. For more information on supported hardware, please refer to the Madwifi's compatibility list [34]. It provides an interfaces for dynamic control of many physical, link, and MAC layer parameters without interrupting ongoing network traffic. In our testbed, we use Netgear WAB501 wireless cards which are based on Atheros AR5001X+ chipset and compatible with the Madwifi driver. For new wireless cards, as long as they are compatible

[3] CLOCK_TICK_RATE

with Madwifi driver they can be used in our platform seamlessly. We extended the existing Madwifi driver to support link virtualization of our platform to achieve a per-packet selection of the physical, link, and MAC layer parameters, link quality information gathering, and nodes synchronization. Due to the FCC regulation on transmission power, the modified driver only supports two power-levels (out of the possible three): 50mW (17dbm) and 5mW (7dbm) [20]. The Netgear WAB501 wireless card supports both 802.11a and 802.11b. In 802.11a mode, the platform allows the protocols under evaluation to switch between 13 channels from 5.15 to 5.825GHz and 8 rates (6, 9, 12, 18, 24, 36, 48 and 54 Mbps). In 802.11b mode, the platform allows the protocols to switch between 11 channels from 2.412 to 2.462GHz and 4 rates (1, 2, 5.5 and 11Mbps).

Platform Infrastructure: The platform is implemented within the Click environment [14]. Click is a novel environment for building flexible and configurable software-routers. It provides a rich collection of modules called elements, and each element realizes one aspect of the router's behavior. It takes a configuration file that specifies connections between user-specified elements to build software routers. It also allows users to write their own elements. We develop our DEC platform as a Click element. Several MANET protocols, such as AODV and DSDV, have been implemented within Click. For more information on source code, user manuals and experimental results related to the DEC platform, refer to [51].

4.2 Platform Architecture

Within Click Architecture: The platform is implemented as a Click element [See Figure 3], which coordinates with four other Click elements, `FromHost`, `ToHost`, `FromDevice`, and `ToDevice`, to become a user-space software router. In one direction, the user-space router intercepts the packets originated at user applications from the kernel, processes them, and hands them to the appropriate network interface. In the other direction, the user-space router retrieves incoming packets from the specified network interface, processes them, and either passes them to the network interface for forwarding, or injects them into the kernel to be delivered to local applications.

- **FromHost:** intercepts host application packets from the kernel and puts them in the platform for processing.
- **ToHost:** injects incoming packets destined to the host into the kernel and the kernel delivers them to applications.
- **FromDevice:** retrieves incoming packets from the given network interface, and put them in the platform for processing.
- **ToDevice:** transmits outgoing packets through the given network interface.

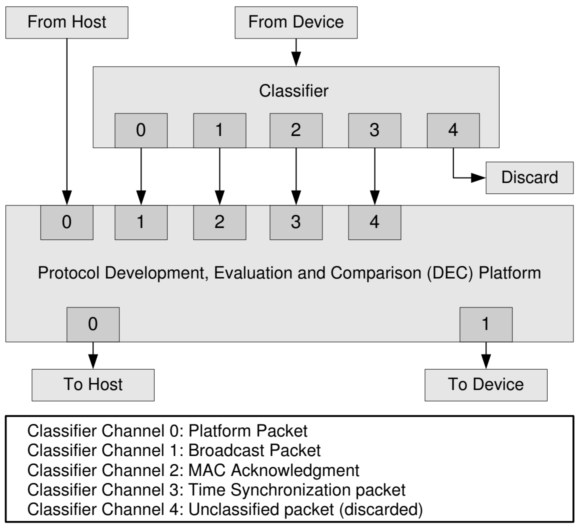
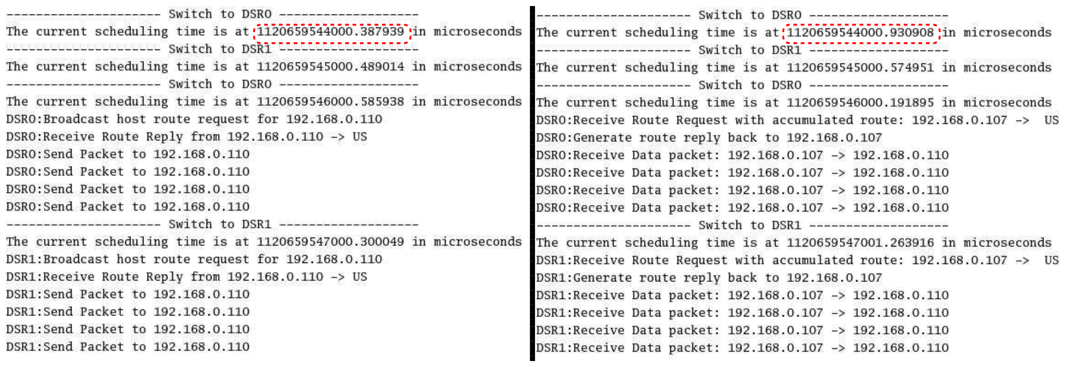

From Host
From Device
Classifier
0
1
2
3
4
Discard
0
1
2
3
4
Protocol Development, Evaluation and Comparison (DEC) Platform
0
1
To Host
To Device
Classifier Channel 0: Platform Packet
Classifier Channel 1: Broadcast Packet
Classifier Channel 2: MAC Acknowledgment
Classifier Channel 3: Time Synchronization packet
Classifier Channel 4: Unclassified packet (discarded)

Fig. 3. The platform within Click architecture.

Platform Internal Structure: From the external point of view, the platform acts as a normal user-space router. Internally, it is both a packet multiplexer and a demultiplexer. On one hand, it collects processed packets from all the protocols and either forwards them to remote sites or passes them down into the kernel. On the other hand, it receives raw packets from outside, either from the kernel or from remote nodes, and dispatches them to appropriate protocols. The platform consists of several components: *Network-Time Synchronization*, *Timeslot Synchronization*, *Time Virtualization*, *Protocol Scheduler*, *Link Virtualization*, and *MAC Acknowledgment* [See Figure 2]. Each component plays a significant role in providing a fair and accurate evaluation of multiple protocols. The platform extends into the Linux driver to collect low-level information about the underlying physical and MAC layers and provide tight time synchronization.

4.3 Time-related Components

In the testbed, each node runs an instance of the platform with multiple protocols under evaluation. The platform schedules protocols under evaluation in a TDMA-like fashion. Each protocol is periodically activated over a fixed-length timeslot. In order to achieve a fair and accurate evaluation of protocols, all the testbed nodes must be synchronized both at the system-time level and the timeslot level. In other words, the same virtual protocol on different nodes must wake-up at the same physical time. Furthermore, the protocol scheduling should not affect the behavior of the protocols, that is, the protocol scheduling must be transparent. Thus, it is imperative for each protocol instance to have access to an independent virtual time clock. Our platform allows each protocol instance to have its own virtual time clock such that the protocol itself only sees a continuous virtual time that is indeed fragmented over physical time intervals.



```
--------------------- Switch to DSR0 -------------------
The current scheduling time is at 1120659544000.387939 in microseconds
--------------------- Switch to DSR1 -------------------
The current scheduling time is at 1120659545000.489014 in microseconds
--------------------- Switch to DSR0 -------------------
The current scheduling time is at 1120659546000.585938 in microseconds
DSR0:Broadcast host route request for 192.168.0.110
DSR0:Receive Route Reply from 192.168.0.110 -> US
DSR0:Send Packet to 192.168.0.110
DSR0:Send Packet to 192.168.0.110
DSR0:Send Packet to 192.168.0.110
DSR0:Send Packet to 192.168.0.110
--------------------- Switch to DSR1 -------------------
The current scheduling time is at 1120659547000.300049 in microseconds
DSR1:Broadcast host route request for 192.168.0.110
DSR1:Receive Route Reply from 192.168.0.110 -> US
DSR1:Send Packet to 192.168.0.110
DSR1:Send Packet to 192.168.0.110
DSR1:Send Packet to 192.168.0.110
DSR1:Send Packet to 192.168.0.110
```

```
--------------------- Switch to DSR0 -------------------
The current scheduling time is at 1120659544000.930908 in microseconds
--------------------- Switch to DSR1 -------------------
The current scheduling time is at 1120659545000.574951 in microseconds
--------------------- Switch to DSR0 -------------------
The current scheduling time is at 1120659546000.191895 in microseconds
DSR0:Receive Route Request with accumulated route: 192.168.0.107 ->  US
DSR0:Generate route reply back to 192.168.0.107
DSR0:Receive Data packet: 192.168.0.107 -> 192.168.0.110
DSR0:Receive Data packet: 192.168.0.107 -> 192.168.0.110
DSR0:Receive Data packet: 192.168.0.107 -> 192.168.0.110
DSR0:Receive Data packet: 192.168.0.107 -> 192.168.0.110
--------------------- Switch to DSR1 -------------------
The current scheduling time is at 1120659547001.263916 in microseconds
DSR1:Receive Route Request with accumulated route: 192.168.0.107 ->  US
DSR1:Generate route reply back to 192.168.0.107
DSR1:Receive Data packet: 192.168.0.107 -> 192.168.0.110
DSR1:Receive Data packet: 192.168.0.107 -> 192.168.0.110
DSR1:Receive Data packet: 192.168.0.107 -> 192.168.0.110
DSR1:Receive Data packet: 192.168.0.107 -> 192.168.0.110
```

Fig. 4. The left side corresponds to the log of a sender node and the right corresponds to the log of the destination node. The log shows the nodes synchronization and packets exchanged. The circled numbers indicate the time in milliseconds on the two nodes when the platform switches protocols.

4.3.1 Network-Time Synchronization

Platform requirement:

Time synchronization between distributed nodes is critical to the platform to avoid neighboring (i.e., interfering) nodes from running non-matching virtual protocols. Ideally, all the nodes within the system should be perfectly synchronized, but in MANETs, it is difficult to achieve a precise synchronization. Plus, there is an increase in overhead associated with implementing a tighter synchronization. Therefore, synchronization is a tradeoff between overhead and accuracy.

In our platform the required accuracy for time synchronization is dependent on the duration of the timeslot. To limit the impact of the nodes clock asynchrony, we set our target asynchrony upper bound to be 1 percent of the timeslot duration. On the other hand, the timeslot duration has to be large enough to comfortably allow the hardware and software to (1) reconfigure the network stack parameters (e.g., frequency selection, power setting), (2) switch to the next virtual protocol, and (3) finish transmitting at least one packet with possible retransmissions and an acknowledgement within the timeslot. The time to reconfigure the physical layer parameters depends on the specific hardware implementation. It varies from several hundred microseconds to tens of milliseconds [38]. The software switch of virtual protocols takes only a few

CPU cycles in our platform. This is a negligible time delay given the speed of today's laptops. The duration of transmitting one packet varies significantly depending on the packet length, the transmission rate, the medium access time, and the number of retransmissions. As many of these values are undetermined and difficult to estimate, we chose by design a relatively large lower-bound on the duration of a timeslot to run experiments in our platform. This lower bound value is 100 milliseconds and is large enough to accommodate protocols unpredictability. At the same time, it is small enough to not lose the effectiveness and fairness of the evaluation. Therefore, this sets our target on upper bound to clock asynchrony to be 1 millisecond. In the platform evaluation, Section 5, we show that our light-weight time synchronization protocol achieves an accuracy well below this target value. The single-hop neighbors are synchronized below $15\mu s$ in our current testbed. Even in a network with several hops, the accumulative error is still within the design target.

Existing schemes in time synchronization:

Network time protocol (NTP) [37] is widely used for the Internet time synchronization. But this protocol is designed for the wired network. It is also complex and computationally intensive while a light-weight time synchronization protocol is necessary for our platform to achieve its goal. Other techniques such as the NIST's short wave radio station (WWV/WWVB) [7] and the Global Position System (GPS) [25,40] achieve global time synchronization but they rely on dedicated external devices which adds an extra cost (possibly unnecessary) to the platform.

Significant research has been done in wireless multi-hop ad hoc networks and sensor networks [5,48,56]. Unlike applications using GPS/WWV/WWVB to achieve a global time synchronization, most applications in wireless ad hoc networks and sensor networks only require local time synchronization. So does our platform. Several protocols have been proposed: Reference Broadcast Synchronization (RBS) [18], Timing-sync Protocol for Sensor Networks (TPSN) [21], Flooding Time Synchronization Protocol (FTSP) [36]. These protocols can reach a synchronization accuracy of a few microseconds to a few tens of microseconds depending on the underlying hardware and support OS. It is worth noting that much better clock synchronization can be achieved in sensor networks in comparison with ad hoc networks. First, this is due to the absence of real time support in the used operating system of ad hoc nodes (mainly linux or Microsoft Windows). And second, it is because of the loose coupling between the communication chipset and the synchronization software. The above mentioned protocols either involve complex message exchanging or require the establishment of hierarchical structures. These constraints do not match our platform's architecture and system requirement. Our synchronization period has to be limited between timeslots and finished before the next timeslot starts. Our protocol is similar to FTSP. However, FTSP also relies on a hierarchical structure starting at the root, which makes it vulnerable to a single point of failure. Although, FTSP provides a root recovery algorithm, for the ease of implementation, which is an important feature to our platform, we chose not to adopt the hierarchical structure and instead use a peer-to-peer mechanism for time synchronization. Our evaluation results show that our synchronization protocol does not compromise accuracy. Its performance is comparable to the existing protocols, and well exceeds the needs of our platform.

Platform time synchronization protocol:

The time synchronization protocol used in the platform consists of two phases:

Phase I: Initialization. All the mobile nodes first synchronize to the system time of one of the nodes (called master). They must be within the master's broadcasting range to estimate their clock's drift-rate and offset to the master's clock.

The clocks' drift-rate varies over long periods of time due to the clock aging and temperature, but tends to be stable in a fairly short period of time (few experiments). We verified this by monitoring the time difference between two nodes. The clock drift turns out to be linear (represented by the cross-points straight line of Figure 5). It implies that the two nodes' clocks are stably running at slightly different rates. This is observed for a typical evaluation experiment of several hours. Even for experiments lasting several days, the clocks' drift-rate can be estimated once in a while, if the clocks' rates have non-negligible change. We also note that although the clocks drift is linear (as shown on Figure 5), the instantaneous measurement values do not perfectly lie on the drift-rate line. In our experiments, we found that this deviation (error) has a zero-average, and 93% of the time, is within the $[-7.5, 7.5]$ interval. Figure 6, shows the histogram of the deviation. One explanation of this inaccuracy is the inability of the non-real time operating system (i.e., linux) to make the reading at the specified instant of time. This inaccuracy can be mitigated by taking a sufficiently large number of measurements to estimate the clock drift-rate. In this phase, the master first broadcasts a series of reference packets and then each slave computes it's clock's drift-rate respect to the master's clock using a linear regression.

To find out the initial clock offset, the master needs to broadcast its timestamp to all the slave nodes. But for each slave node to make the right adjustment, they must know the communication latency. This latency is composed of the processing time for transmitting and receiving messages and the propagation time. In our platform the processing time is deterministic and according to our measurement this value is approximately $155\mu s$. The propagation time, however, is not a constant as it depends on the distance between the transmitter and the receiver. Under similar assumptions made by many existing synchronization protocols [18], we consider this value to be negligible. This is because the propagation speed of radio frequency signals is close to the free-space speed of light *c*. For typical MANET links covering tens of meters, this propagation time varies only in nanoseconds which is negligible with respect to the precision required by the platform.

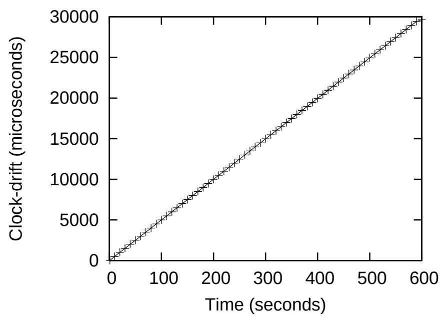


Fig. 5. The clock difference of two clocks over the period of 10 minutes.

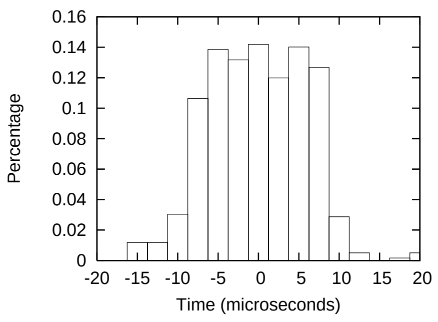


Fig. 6. Histogram of deviation from the estimated drift-rate line. 93% of the time within interval $[-7.5, 7.5]$.

Phase II: Re-synchronization. Due to the inaccuracy of the clock drift and offset estimation, a synchronization error will accumulate over time. To prevent this asynchrony from having a non-negligible impact, mobile nodes need to regularly re-synchronize with their neighbors by broadcasting the synchronization packets in a given time window. In our platform, the user can easily adjust the frequency of the time synchronization messages being sent according to the synchronization accuracy he/she to achieve. By default, the average synchronization period is set to 1 second which causes limited overhead while still achieving good synchronization accuracy. To avoid collisions each node randomly picks a instant within the synchronization window to broadcast a packet with its timestamp. Since the synchronization packets are broadcast, they are not delayed by the IEEE802.11 MAC RTS/CTS or backoff mechanisms. Therefore, the received timestamps have better freshness characteristics.

On the receiver side, when the wireless card driver receives a time synchronization packet, it extracts the timestamp out of the packet, adjusts it to take in account the estimated communication latency, retrieves its own current timestamp, and uses the average of the two to set the system time.

In our network-time synchronization scheme, the synchronization packet originates within the user-space platform, while the actual time synchronization operations are performed within the driver [See Figure 2]. Since, the latter is implemented at the driver level, it minimizes the delay between the timestamp generation at the source and time update at the receiver side. Our experiments confirm that the system time difference between adjacent nodes can be limited to within 15μs when the average synchronization period is set to 1 second. A run-time snapshot of the synchronization process is demonstrated in Figure 4.

4.3.2 Timeslot Synchronization

When the system time of the testbed nodes is synchronized, the nodes must also agree on when to start a new timeslot, and which protocol to activate. The platform protocol scheduler determines which protocol to activate for each timeslot according to a deterministic rule (partially defined by the user for each experiment via a configuration file), which is discussed in section 4.4. That rule is solely based upon the knowledge of the number of protocols and their identification along with the length of the timeslot. Such information is common to all the nodes. Thus, it allows the testbed to synchronize the timeslot and schedule protocols without exchanging any information. The scheduling pattern simply repeats over time at all nodes.

Our platform also considers the issue of overlapping packets. It happens when a packet is initiated in one timeslot but delivered while the destination node has already switched onto the next timeslot. In our platform, this issue is first addressed by maintaining a buffer for each protocol to store those overlapping packets instead of mis-delivering them to a wrong target protocol. Since only the active protocol has access to the system resources and network bandwidth, the overlapping packets will be held for delivery until the target protocol becomes active again. Another way to resolve this problem is by adding a guard time between the switching of protocols. This allows the transmission of current protocol packet to complete before the platform switches onto the next protocol. However, this approach causes another issue of protocols deviating from their normal behavior. For example, less contention among the participating nodes during the guard time. The bottom line is that there is at most one overlapping packet at the boundary. Thus, the problem of overlapping packets should have a relatively small impact on the performance if there are a large number of packets are transmitted in a timeslot.

4.3.3 Time Virtualization

Each protocol in the platform has its own virtual time clock, which is continuous from the protocol's point of view. Assume VT_X is the virtual time for protocol instance X. Then, in the platform, VT_X freezes whenever the current timeslot is running other protocols. In other words, VT_X freezes whenever protocol X is not currently active. VT_X resumes once the protocol X becomes active again. For instance, if a protocol sets up a timer to expire in 20 milliseconds and the current timeslot terminates in 5 milliseconds, then the timer will fire 15 milliseconds after the reactivation of the considered protocol. An example of slotted virtual time and physical time is shown in Figure 7.

The platform keeps track of the virtual time for each protocol. The virtual time advances only when the associated protocol becomes active. The protocols can be perfectly interleaved as long as they use the virtual time methods provided by the platform to manage their time and set their timers.

The length of each timeslot (TS) is a critical parameter of the platform. Intuitively, this length should neither be too short because of the overhead, time synchronization inaccuracies, risk of packets overlapping and interfering with the TS switching-time. The length should also not be too long otherwise the macro-characteristics of the environment will change (less stable) during a longer timeslot and therefore the fairness and accuracy of the evaluation result will be compromised. Refer to Section 5 for detailed discussion and our experiment result on the optimal value for the timeslot duration.

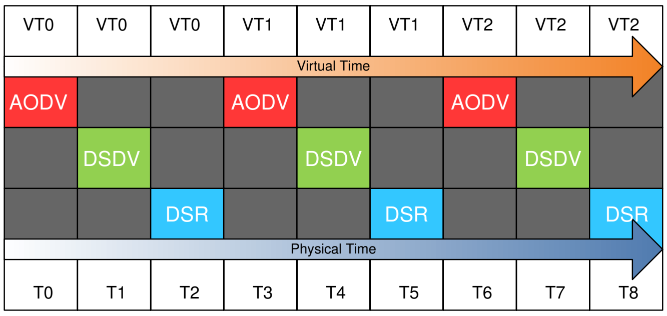


Fig. 7. Protocols scheduling and virtual time vs. physical time.

4.4 Protocol Scheduler

The Protocol scheduler decides which protocol to activate for each timeslot. For each experiment, the user specifies which protocols to compare. The scheduler uses a rule that requires the product of the timeslot duration and the number of protocols, to be either a multiple of a second or a perfect divider of a second. Thus, the protocol scheduling pattern will repeat every N seconds, where N is the smallest positive integer that is a multiple of the product value from the rule. For instance, if the number of protocols is 3 and the timeslot is 400 milliseconds, the scheduling pattern repeats every $N = 6$ seconds. The platform mandates that the first protocol is always scheduled to be active at a system time that is a multiple of N seconds. When the platform starts, it loops on the current system time until the system time advances to become a multiple of N seconds. Then it activates the protocol scheduler, which in turn wakes up the first protocol. The protocol scheduler always wakes up at the beginning of each timeslot, puts the current protocol into sleep mode and activates the next protocol.

The protocol scheduler is driven by a system timer, whose precision is determined by the *tic time* (1 millisecond). A tiny difference might happen between the protocol scheduler's actual wake-up time and the expected wake-up time. In our testbed, this difference is about 1 microsecond. But the protocol scheduler continuously corrects those tiny drifts to avoid the accumulation of such drifts.

4.5 Link Virtualization

The platform provides each protocol instance with a *virtual link*. The same protocol instance runs on all the nodes and is scheduled at all nodes at the same time. So, the platform makes sure that the packet sent by a protocol instance is received by its corresponding instance on the remote nodes. This is achieved by introducing an additional layer, that we call a virtualization layer, into the traditional IP architecture. The virtualization layer takes care of the scheduling of protocol-instances, queues multiplexing/demultiplexing, and segregation of the instances. The platform inserts the virtualization header into the packet received from the protocol before passing it to the network interface. On the remote nodes, the platform removes this header before dispatching it to the appropriate protocol. This process is totally transparent to the protocol-instances running in the platform. The platform header is only 8 bytes in length, thus introducing a very small (compared to the size of the network payload) overload to the network traffic [See Figure 8].

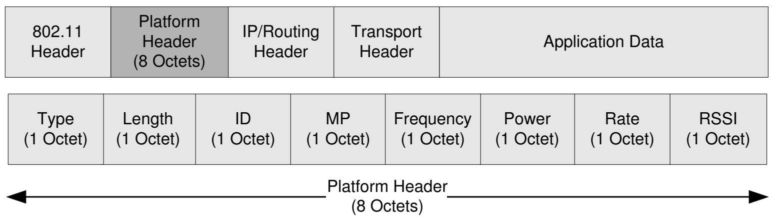


Fig. 8. Packet header.

The platform header contains eight fields, Type, Length, ID, MP, Frequency, Power, Rate, and RSSI:

- **Type** field contains the type of the next frame.
- **Length** field contains the size of the platform header.
- **ID** field identifies the protocol that the packet belongs to.
- **MP** field contains MAC Control Parameters: RTS/CTS, fragmentation, number of retransmissions.

- **Frequency** field selects the RF frequency channel on which the wireless NIC is working.
- **Power** field selects the transmission power.
- **Rate** field selects the transmission rate.
- **RSSI** field is updated at the receiver node to include the RSSI value that the packet was received at.

This virtual link allows users to send data packets at the frequency/power/rate/ modulation specified by the protocols. The actual execution of these parameters is carried out at the driver level. When the driver receives a packet from the platform, it extracts the frequency, power and rate specification and instructs the firmware to transmit the packet at those specified values. When the driver receives an incoming packet, which contains the platform header, it collects the RSSI value for the packet and writes it in the RSSI field of the platform header in the packet.

4.6 MAC Acknowledgment and Protocol Performance Monitor

MAC acknowledgments are not only useful to determine if a link is broken but also to estimate the network medium and channel state. Whenever a packet is successfully sent from the platform and acknowledged (at IEEE 802.11 level) or if the maximum retransmission threshold is reached and the packet is dropped, a MAC acknowledgment is generated within the driver and passed up to the platform. The MAC acknowledgment contains the information about how many retransmissions the packet attempted, at which frequency channel, power level, and rate the packet was sent, and which protocol the packet belongs to. The MAC acknowledgment also helps the cross-layer protocols to estimate the link quality. The protocol performance-monitor running in the platform logs this information for each packet, which is eventually used to evaluate the performance of each protocol (in terms of resource usage such as, bandwidth, and energy/power).

4.7 Developing New Protocols for the Platform

The platform allows a transparent development and integration of new protocols. The protocols developed for the platform can be used in other environments such as network simulators. Likewise, the protocol implementations from other environment can also be ported onto the platform as long as they respect a simple interface described below.

The platform is implemented in C++. A new protocol has to be declared as a C++ class and be inherited from the super class `PhysicalProto`. The default implementation of `PhysicalProto` registers the derived protocol with the platform and provides the protocol with the interfaces to receive packets, deliver packets and access virtual time. Subclasses of PhysicalProto are easy to write. Essentially there are three virtual methods that need to be overridden.

```
 virtual VOID  Process() = 0;
 virtual VOID  ReceiveNetPkt(Packet* p, PROTO_RX  rx) = 0;
 virtual VOID  ReceiveHostPkt(Packet* p);
```

The platform notifies the custom protocol of incoming packets from other hosts through the method `ReceiveNetPkt()`. The method `ReceiveHostPkt()` allows the custom protocol to receive packets originated within the local host. The method `Process()` can be used to process pending packets and do housekeeping tasks. The implementation of these methods is fairly simple. In most cases, they are just forwarding the function calls. For example, the DSR protocol we implemented for the platform only overrides the above-mentioned three methods. The methods `ReceiveNetPkt()` and `ReceiveHostPkt()` both have only one statement inside them, which simply forwards the call to the packet-processing function. The `Process()` method calls another method that polls the sub-modules for internal processing. It is easy to see that these methods are just wrappers to integrate the developers protocol into the platform.

The protocol must use the virtual time provided by the platform instead of the real system-time and needs to notify the platform every time it has packets to transmit. Then, the platform provides virtual time and packet delivery services to the protocol through several protected methods defined in the class `PhysicalProto`:

```
VOID   __SendPkt(Packet* p, UINT freq, UINT power,
              UINT rate, BOOL monitored = FALSE);
VOID   __SendPkt(Packet* p, UINT freq, UINT power,
              BOOL monitored = FALSE); // use fixed or auto-rate
VOID   __IndicatePkt(Packet* p);
VOID   __SetTimer(CALLBACK func, VOID* arg, UINT t);
DOUBLE  __GetCurrTimeInMs();
```

The method `__SendPkt()` sends out a packet to remote hosts. The protocol can either specify the desired frequency channel, power and rate or just frequency channel and power and let the driver use the default fixed/auto-rate mode. `__IndicatePkt()` indicates a packet to the local host. The method `__SetTimer()` sets up a timer in the virtual time space. The protocol can obtain the current virtual timestamp by calling `__GetCurrTimeInMs()`. All these methods are abstract and the implementation details are hidden from the cross-layer protocols. This way, the protocol implementation is independent of the platform and can be used in other environments.

Existing cross-layer protocols implemented within other platforms, for example in network simulators, can be ported into our platform with a small set of modifications, similar to the ones illustrated above. If the protocol is implemented as a C++ class, one can declare it as a subclass of `PhysicalProto`, override the necessary public interfaces defined in `PhysicalProto`, replace the physical time with virtual time, and use the platform to deliver the packets. Plugging a new protocol into the platform for evaluation is straightforward as well. It only requires a user to declare an instance of the new protocol inside the initialization method of the `EvalPlatform` class:

```
new Proto (this, __hostIp, __hostMac);
```

5 Platform Evaluation

We first evaluate the network-time synchronization protocol used in our platform. Then, we evaluate the capability of our platform to fairly compare two protocols.

5.1 Evaluation of Network-Time Synchronization

The platform relies on the information delivered by broadcast packets in getting the nodes to synchronize with each other in the network. Therefore, the higher frequency of these packets being broadcasted implies the better accuracy of the synchronization. However, an unnecessarily high accuracy of the synchronization might result in a significant communication and energy overhead to the platform. We must optimize the tradeoff that time synchronization introduces in terms of the overhead and the synchronization precision.

Our experiments show that with an average re-synchronization period of 1 second, which does not add significant cost to the system, our synchronization protocol achieves a very good synchronization accuracy of well under the design target ($1ms$). We measured the clock difference between two adjacent nodes for a time span of 30 minutes. Figure 9 shows the histogram of the clock differences. As we can see, the majority of the clock differences over time are between 5 to 10 μs. From Figures 10, we find that over 95% of the time, the clock difference is bounded by $15\mu s$. We also notice that there are a few clock asynchrony jitters above $45\mu s$. This is mainly due to the fact that the operating system we are using is not a realtime system and at times, it might allocate the CPU resources to deal with an unexpected interrupt (which would temporarily mask the wireless card interrupts) thereby causing a jitter. Porting the platform to a realtime operating system would better guarantee the prioritized execution of time critical components.

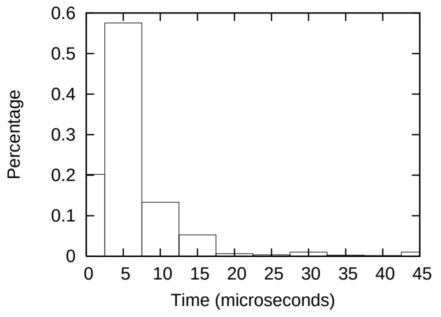


Fig. 9. The histogram of the clock difference of two adjacent nodes with re-synchronization period 1s.

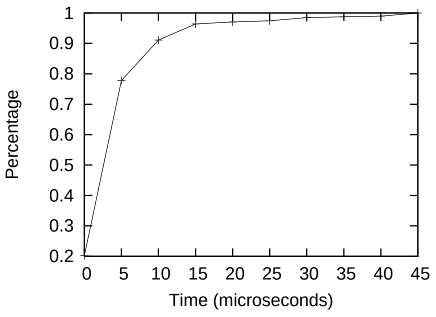


Fig. 10. The accumulative percentage of clock difference of two adjacent nodes with re-synchronization period 1s.

5.2 Evaluation of the Platform Protocol Comparison Capability

We run an extensive set of experiments to validate the DEC platform capability in fairly comparing two protocols (over 100 experiments). We run two instances of the same protocol in our platform and showed that the instantaneous performance (number of packets received per timeslot, transmission energy, etc.) is highly correlated. We picked DSR as our test protocol. When we compared the separate runs of DSR (non-interleaved) following the same mobility patterns with minimal human perturbation and found a much lower correlation.

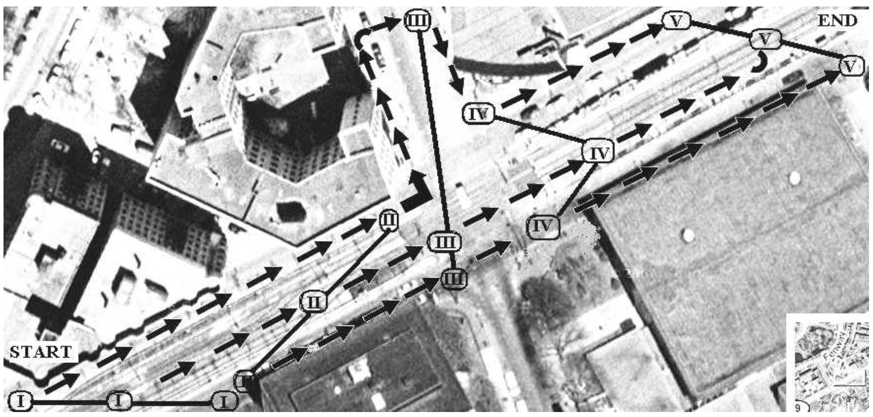


Fig. 11. Mobility pattern 1. In/Around Northeastern University, Boston.

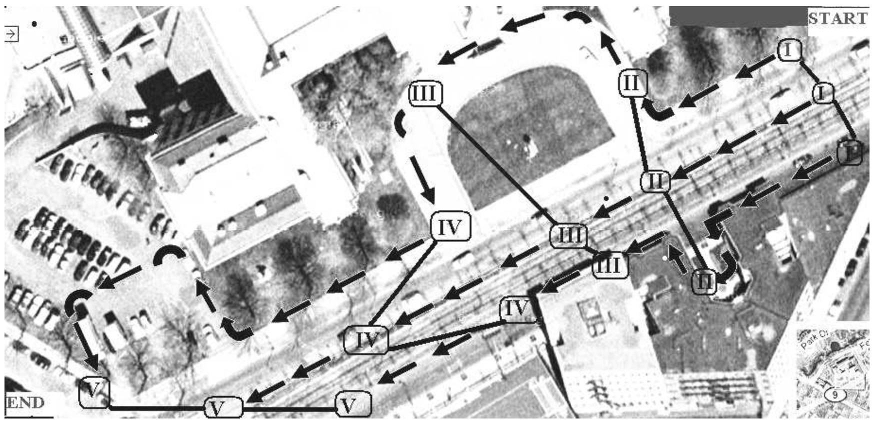


Fig. 12. Mobility pattern 2. Around Museum of Fine Arts, Boston.

5.2.1 Experimental Setup

Scenarios: We choose three different scenarios to run our experiments, two outdoor and one indoor. Our setup aims at distinguishing the platform's innovative approach and establishing its main inspiration: fair and credible comparison of MANET protocols. Figures 11 and 12 show two outdoor mobility patterns that span areas of around 500 meters × 300 meters. Pattern 1 features a typical (busy streets with buses and cars) MANET scenario around Northeastern University, Boston. Pattern 2 features another typical (busy streets with buses and cars along with the trains passing by) MANET scenario around the Museum of Fine Arts, Boston. We choose the Computer Science building of our school as the sit to run the indoor experiments, that consisted of a predefined trajectory around a big lab with no line of sight.

Experiment: *(Outdoor)* We have three people, each carrying a laptop, walking in the pattern defined in Figures 11 and 12 that establishes a multi-hop communication. *(Indoor)* We have two people, walking around a (13 meters × 13 meters) lab carrying two laptops with no line of sight. The third laptop is stationary sitting in a lab desk inside the room, again establishing a multi-hop communication (when the source and destination are not within a *reachable* distance). The latter of the setups has significant channel interference due to the presence of foreign wireless networks and microwave ovens. Both indoor and outdoor interleaved experiments are run for 10 minutes each and the non-interleaved ones are run for 5 minutes each.

Parameters and Analysis: In all scenarios, there is a single source and a single destination with an intermediate node. We inject 50 or 10 packets per second into the platform where each packet is 512 bytes in size. Our platform does not incorporate any coordination with the fourth or higher layer protocols. Typically, we ran 10 interleaved and 8 non-interleaved experiments for each scenario (two outdoor and one indoor). The experiments covered cases where the interleaving period spans 100ms, 250ms, 500ms, 1000ms, 2000ms

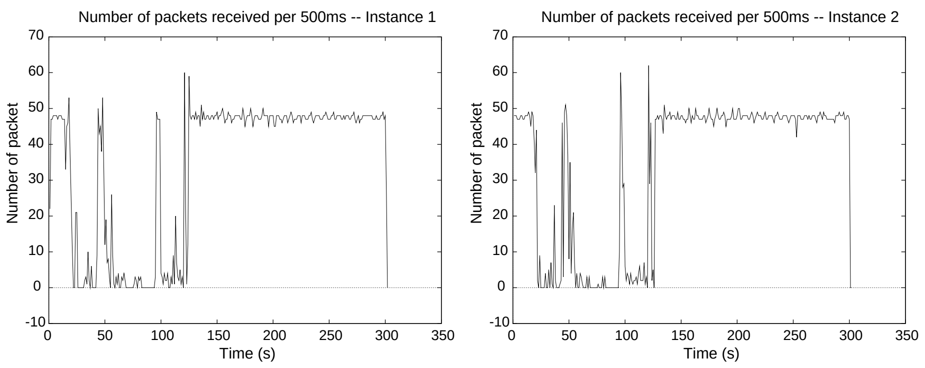


Fig. 13. Number of packets received by interleaved instances of DSR protocol in a typical experiment with mobility pattern 1.(Figure 11)

and 3000ms in order to compare the correlation and thus identify the best average interleaving period corresponding to the best correlation for the environment we chose to run experiments in. With non-interleaved runs, we tried our best to accurately reproduce the same mobility pattern as of the previous run.

5.2.2 Results

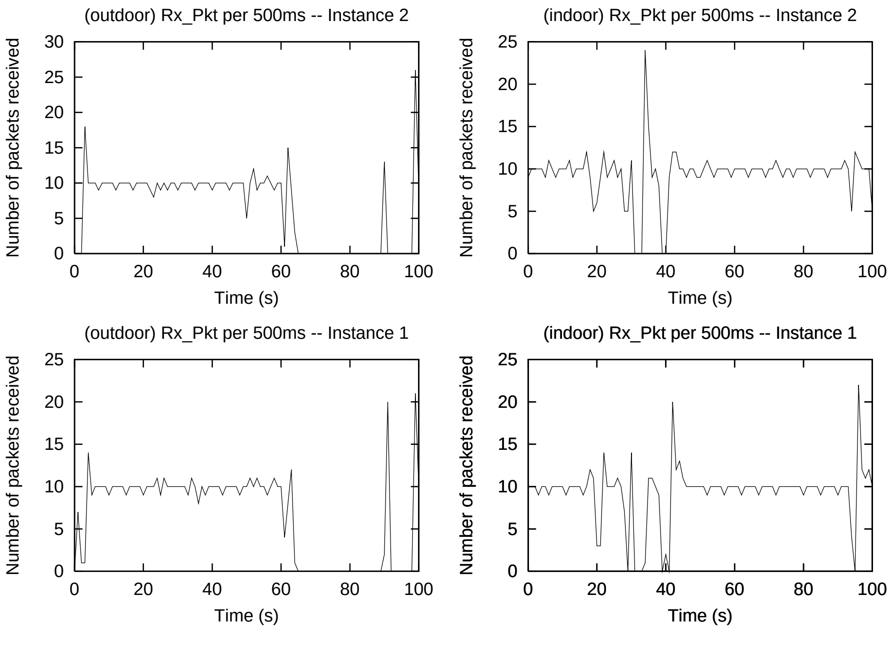


Fig. 14. Number of packets received by interleaved instances of DSR protocol in a typical experiment with (LEFT) mobility pattern 2 and (RIGHT) indoor scenario: CCIS building - Northeastern University.

Correlation of Instantaneous Throughput, Loss Rate, and Energy: Figures 13, 14 and 15 show typical graphs of the number of received packets vs. timeslots. Figures 13, and 14 represent the interleaved case, which shows a strong similarity in received packets per timeslot. Figure 15 represents the non-interleaved case where we run two instances of DSR protocols individually at a 5 minutes time interval with exact mobility pattern and least human perturbation. The non-interleaved case exhibits evident dissimilarity between the received packets pattern. The correlation between the number of packets received by two interleaved instances computed over all the experiments was 0.89 (indoor) and 0.85 (outdoor). The average pairwise correlation between non-interleaved runs was 0.38 (indoor) and 0.28 (outdoor).

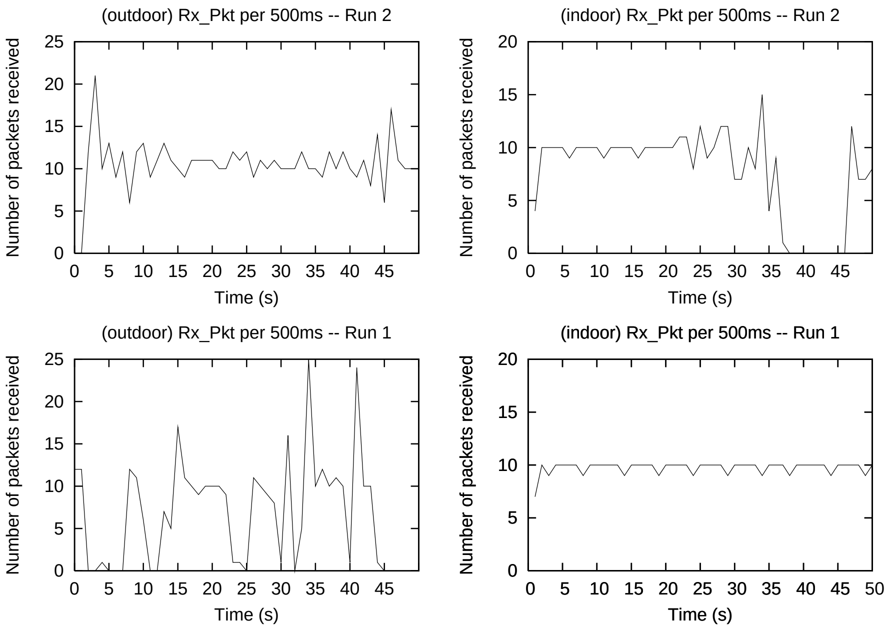


Fig. 15. Number of packets received by non-interleaved runs (ran at 5 minutes interval) of DSR protocol.(Left::Outdoor) and (Right::Indoor)

Figure 16 is the average histogram of difference between the number of packets received by two DSR instances. We notice that a packet difference of 1 or 2 is more common in the interleaved case than in the non-interleaved case. Consequently, bigger discrepancies are rather frequent in separate runs (non-interleaved case) of DSR protocol instances[4]. In particular, the 10 packets difference is very frequent among non-interleaved runs of protocol instances. This is because the excerpt is from the experiment set where we were sending 10 packets per timeslot into the platform. So, the discrepancy indicates that one link is broken while another is still active in the test-bed.

The platform also provides information about the number of retransmissions, rate, and transmission power. We aggregate this information and compared the energy expenditure between the interleaved protocol instances and the

[4] The ”zero” difference bar is almost of the same height in both interleaved and non-interleaved instance because in our setup, link breakage and perfect links occur periodically in all scenarios.

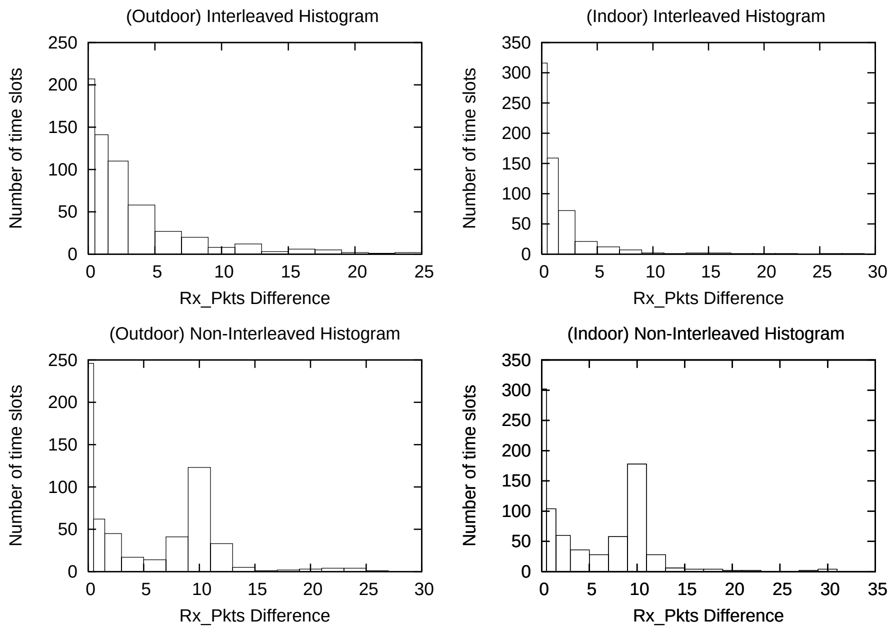


Fig. 16. The average of differences in number of received packets between two instances of protocol (LEFT::Outdoor) and (RIGHT::Indoor), (TOP::Interleaved) and (BOTTOM::Non-Interleaved).

separately run protocol instances. We observe that the separately run protocols exhibit a very different energy pattern while the instantaneous energy expenditure of the interleaved runs is relatively similar. See Figure 17 for the energy expenditure in a typical interleaved and a separate-run experiment. For the complete experimentation results and detailed graphs, the reader is referred to [51].

Impact of Interleaving Period and Averaging Window: In order to identify the best interleaving period, we carried the experiments with various interleaving periods. Figure 18 shows the varying correlations when different interleaving periods were picked for both indoor and outdoor experiments. The demonstrated correlation is between the number of packets received per timeslot by two interleaved protocol instances. We find that 500ms interleaving period gives the best correlation value among 100ms, 250ms, 500ms, 1000ms, 2000ms and 3000ms under the current setup.

We define the averaging window as a time frame (collection of timeslots) that we used to accumulate the number of received packets or other performance measures. Figure 19 shows the correlation variation as per the increase in the averaging window size. The figure indicates that as the averaging-window size gets larger, the correlation gets closer to 1. The average difference in total delivered packets reaches less than 1.5% for the interleaved runs, and over 17% (with high variance) for the non-interleaved runs [51].

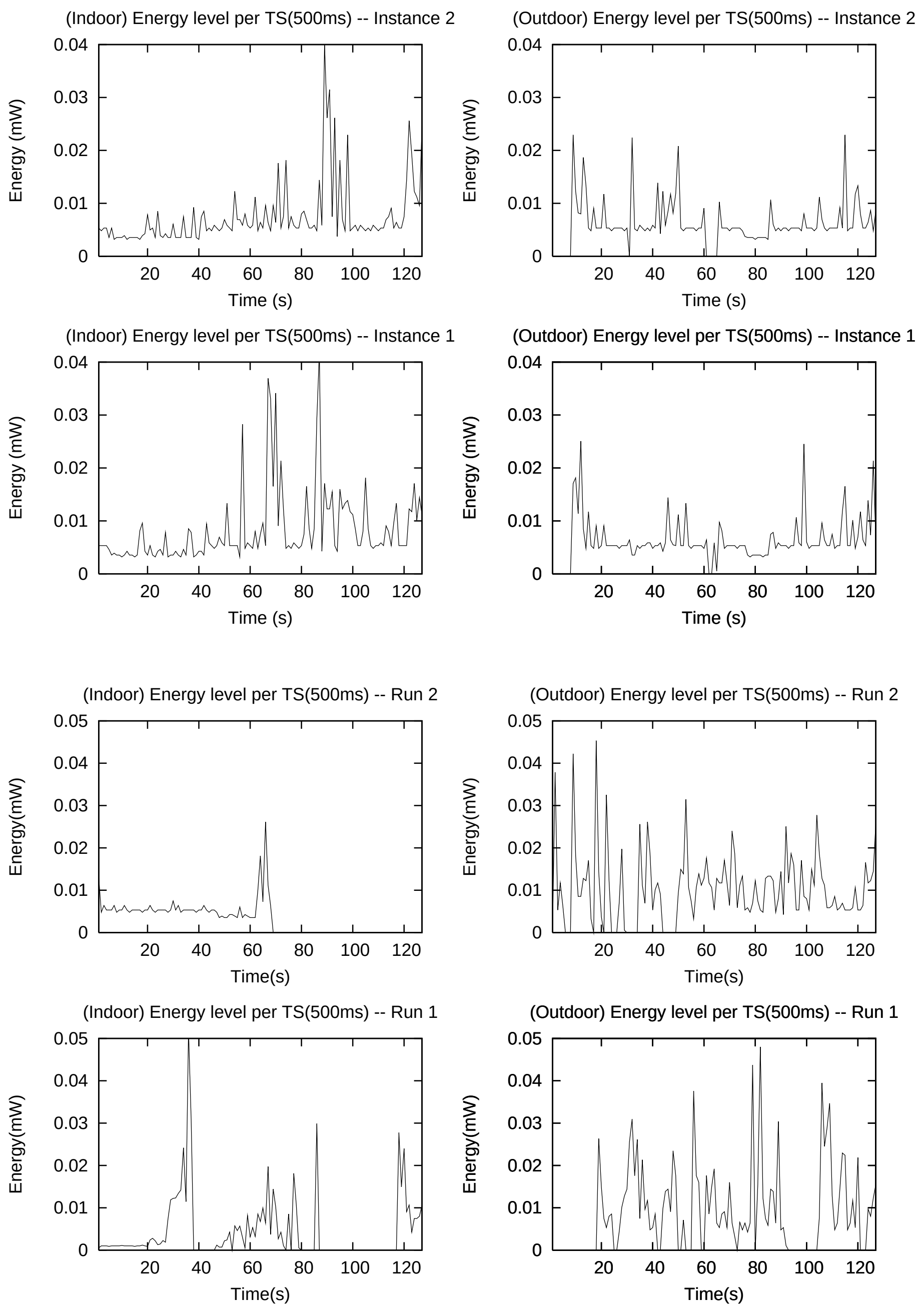


Fig. 17. Transmission energy expenditure between (upper 4 graphs) interleaved and (bottom 4 graphs) non-interleaved runs of DSR protocol.

6 Conclusion

We proposed a platform for the performance evaluation and comparison of multiple protocol stacks under an almost identical physical environment that includes same channel macro-characteristics, topology, and geography. We implemented the framework and demonstrated its validity by showing that the

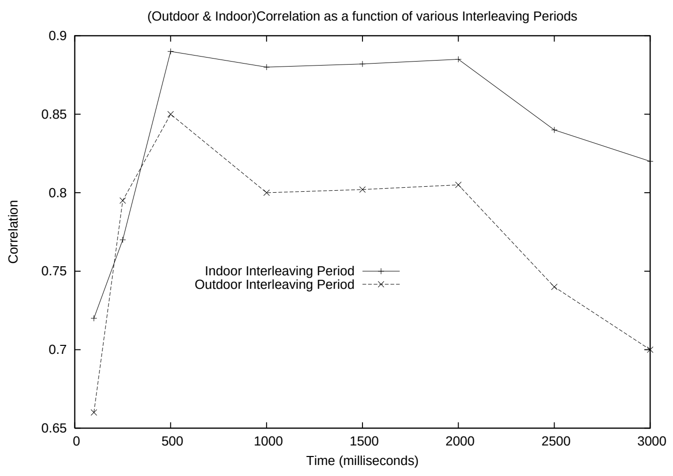


Fig. 18. Average correlation per varying Interleaving period:Graph showing 500ms as the best interleaving period.

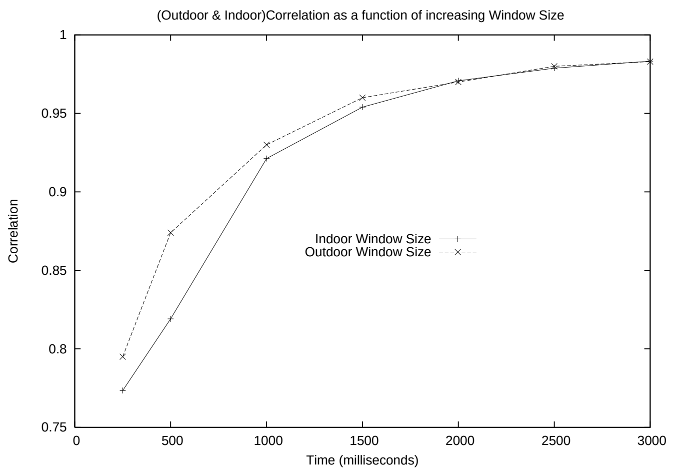


Fig. 19. Average correlation per increase in window size (accumulation of received packets over a time frame).

instantaneous performance of two interleaved DSR instances is highly correlated, while the performance of separately run instances are weakly correlated (even if we did our best to accurately reproduce the same mobility pattern). In addition to allowing a fair comparison of multiple protocols, our framework also provides a set of services to control the physical layer and MAC/Link layer parameters on a per-packet basis(e.g., per packet frequency/power/rate control, fragmentation, and retransmissions threshold).

We plan to port other MANET protocols to the platform in order to compare them within our platform and compare our results with those obtained through simulation. Finally, we plan to develop a set of tools for easy deployment of new protocol stacks over the testbed. With these tools new member nodes of the testbed only need to have a bootstrap daemon to download the configuration files and the added protocol stacks to join an experiment. Source code, documentation, and experimentation results of the current platform are available from [51].

References

[1] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *SIGCOMM '04*, 2004.

[2] I. F. Akyildiz, M. C. Vurah, and O. B. Akan. A cross-layer protocol for wireless sensor network. In *CISS '06: Proceedings of the 40th annual conference on Information Sciences and Systems*, pages 1102–1107, Piscataway, NJ, USA, 2006. IEEE Educational Activities Department.

[3] T. Andel and A. Yasinsac. On the credibility of manet simulations. *IEEE Computer*, 39(7), July 2006.

[4] T. Arildsen and F. H. P. Fitzek. Cross layer protocol design for wireless communication. *Mobile Phone Programming*, 6:343–362, June 2007.

[5] U. B. B. Sundararaman and A. Kshemkalyani. Clock synchronization in wireless sensor networks: A survey. *Ad-Hoc Networks*, 3:281–323, May 2005.

[6] S. Banerjee and A. Misra. Minimum energy paths for reliable communication in multi-hop wireless networks. In *MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 146–156, New York, NY, USA, 2002. ACM Press.

[7] R. Beehler. Time/frequency services of the u.s. national bureau of standards and some alternatives for future improvement. *Journal of Electronics and Telecommunications Engineers*, 1981.

[8] C. Bettstetter, H. Hartenstein, and X. Perez-Costa. Stochastic properties of the random waypoint mobility model: epoch length, direction distribution, and cell change rate. In *MSWiM '02*, 2002.

[9] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97, New York, NY, USA, 1998. ACM Press.

[10] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of manet simulators. In *POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 38–43, New York, NY, USA, 2002. ACM Press.

[11] I. D. Chakeres and E. M. Belding-Royer. The utility of hello messages for determining link connectivity. In *Proceedings of the 5th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, 2002.

[12] J. Chen, L. Jia, X. Liu, G. Noubir, and R. Sundaram. Minimum energy accumulative routing in wireless networks. In *Proceedings of IEEE INFOCOM*, 2005.

[13] T. Chu and I. Nikolaidis. On the artifacts of random waypoint simulations. In *First International Workshop on Wired/Wireless Internet Communications (WWIC 2002) in conjunction with the 3rd International Conference on Internet Computing 2002 (IC 2002)*, June 2002.

[14] The click modular router project. http://read.cs.ucla.edu/click/.

[15] D. S. J. D. Couto, D. Aguayo, B. A. Chambers, and R. Morris. Performance of multihop wireless networks: shortest path is not enough. *SIGCOMM Comput. Commun. Rev.*, 33(1), 2003.

[16] J. del Prado Pavon and S. Choi. Link adaptation strategy for ieee 802.11 wlan via received signal strength measurement. In *IEEE International Conference on Communications (ICC'03)*, volume 2, pages 1108–1113, Anchorage, Alaska, USA, May 2003.

[17] S. Doshi, S. Bhandare, and T. X. Brown. An on-demand minimum energy routing protocol for a wireless ad hoc network. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(3):50–66, 2002.

[18] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163, 2002.

[19] Emulab robotic testbed. http://www.emulab.net.

[20] Authorization and use of software defined radios. http://ftp.fcc.gov/Bureaus/Engineering_Technology/Orders/2001/fcc01264.pdf.

[21] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149, 2003.

[22] R. S. Gray, D. Kotz, C. Newport, N. Dubrovsky, A. Fiske, J. Liu, C. Masone, S. McGrath, and Y. Yuan. Outdoor experimental comparison of four ad hoc routing algorithms. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems (MSWiM)*, pages 220–229, 2004.

[23] G. Holland, N. Vaidya, and P. Bahl. A rate-adaptive mac protocol for multi-hop wireless networks. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 236–251, New York, NY, USA, 2001. ACM Press.

[24] Wireless tools for linux. http://www.hpl.hp.com/personal/Jean_Tourrilhes /Linux/Tools.html.

[25] E. D. Kaplan, editor. *Understanding GPS: Principles and Applications.* Artech House, 1996.

[26] T. Kaya, G. Noubir, and A. Yilmaz. Note on discrepancies between simulation and field measurements in ad hoc networks routing protocols. Technical report, Northeastern University, CCIS, 2003.

[27] W. Kiess and M. Mauve. A survey on real-world implementations of mobile ad-hoc networks. *Ad hoc Networks*, 5:324–339, April 2007.

[28] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions Computer Systems (TOCS)*, 18(3), 2000.

[29] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions. In *MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 78–82, New York, NY, USA, 2004. ACM Press.

[30] U. C. Kozat, I. Koutsopoulos, and L. Tassiulas. A framework for cross-layer design of energy-efficient communication with qos provisioning in multi-hop wireless networks. In *Proceedings of IEEE INFOCOM*, 2004.

[31] S. Kurkowski, T. Camp, and M. Colagrosso. Manet simulation studies: the incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(4), October 2005.

[32] G. Lin, G. Noubir, and R. Rajamaran. Mobility models for ad-hoc network simulation. In *Proceedings of IEEE INFOCOM*, 2004.

[33] Official madwifi website. http://madwifi.org/.

[34] Madwifi compatibility list. http://madwifi.org/wiki/Compatibility.

[35] D. A. Maltz, J. Broch, and D. B. Johnson. Lessons from a full-scale multihop wireless ad hoc network testbed. *IEEE Personal Communications*, 8(1), February 2001.

[36] M. Maróti, B. Kusy, G. Simon, and Ákos Lédeczi. The flooding time synchronization protocol. *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, 2004.

[37] D. L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, 1991.

[38] A. Mishra, V. Shrivastava, D. Agrawal, S. Banerjee, and S. Ganguly. Distributed channel management in uncoordinated wireless environments. *MobiCom '06: Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 170–181, 2006.

[39] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[40] U. D. of Defense. Global positioning system standard positioning service, signal specification. 1993.

[41] Opnet modeler. http://www.opnet.com.

[42] Orbit: Open-access research testbed for next-generation wireless networks. http://www.orbit-lab.org.

[43] W. Qian, G. Noubir, X. Liu, and P. Ramachandra. Link virtualization: A framework for simultaneous evaluation of multiple protocols. In *V2V Communications '05: 1st international workshop on Vehicle to Vehicle Communications*, 2005.

[44] D. Qiao, S. Choi, and K. G. Shin. Goodput analysis and link adaptation for ieee 802.11a wireless lans. *IEEE Transactions on Mobile Computing*, 1(4):278–292, 2002.

[45] Qualnet. http://www.qualnet.com/.

[46] C. S. D. W. R. Ramanathan, J. Redi and S. Polit. Ad hoc networking with directional antennas: A complete system solution. *EEE Journal on Selected Areas In Communications*, 23:496–506, March 2005.

[47] A. K. Saha, K. A. To, S. Palchaudhuri, S. Du, and D. B. Johnson. Physical implementation and evaluation of adhoc network routing protocols using unmodified simulation models. In *SIGCOMM ASIA WORKSHOP*. ACM, April 12-14, 2005.

[48] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: a survey. *IEEE Network*, 18:45–50, July 2004.

[49] A. Srinivas and E. Modiano. Minimum energy disjoint path routing in wireless ad-hoc networks. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 122–133, New York, NY, USA, 2003. ACM Press.

[50] M. Takai, J. Martin, and R. Bagrodia. Effects of wireless physical layer modeling in mobile ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, 2001.

[51] Link virtualization framework. http://www.ccs.neu.edu/home/noubir/projects/virtualization.html.

[52] H. Wu, X. Wang, Y. Liu, Q. Zhang, and Z.-L. Zhang. Softmac: layer 2.5 mac for voip support in multi-hop wireless networks. Santa Clara, California, September 2005. IEEE Sensor and Ad Hoc Communications and Networks (SECON).

[53] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *Proceedings of IEEE INFOCOM*, 2003.

[54] J. Yoon, M. Liu, and B. Noble. Sound mobility models. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 205–216, New York, NY, USA, 2003. ACM Press.

[55] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: A library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.

[56] D. Zhou and T.-H. Lai. An accurate and scalable clock synchronization protocol for ieee 802.11-based multihop ad hoc networks. *IEEE Trans. Parallel Distrib. Syst.*, 18(12):1797–1808, 2007.