

# EPEW: An Extended Prototyping Environment for Wireless Mesh Networks

Fehmi Ben Abdesslem<sup>1</sup>, Luigi Iannone<sup>2</sup>, Marcelo Dias de Amorim<sup>1</sup>, Katia Obraczka<sup>3</sup>

<sup>1</sup>UPMC Univ Paris 06    <sup>2</sup>TU Berlin / Deutsche Telekom Laboratories    <sup>3</sup>University of California, Santa Cruz  
 {fehmi,amorim}@rp.lip6.fr    luigi@net.t-labs.tu-berlin.de    katia@cse.ucsc.edu

**Abstract**—This paper introduces EPEW, an environment for rapid prototyping of communication protocols over IEEE 802.11 mesh networks. EPEW provides a software environment that makes prototyping as quick, easy, and effortless as possible, which allows researchers to conduct both functional assessment and performance evaluation as an integral part of the protocol design process. Since EPEW runs on real IEEE 802.11 wireless mesh routers and clients, prototypes can be evaluated and adjusted under realistic conditions. EPEW facilitates prototype development by providing: (i) a set of building blocks that implement common functions needed by a wide range of protocols for wireless mesh networks, and (ii) an API that allows protocol designers to access EPEW primitives.

## I. INTRODUCTION

There is no doubt that ubiquitous access to wireless communication has had a tremendous impact on today’s society. The mote “connected any time anywhere” is fast becoming a reality. The impact of wireless communication technology has manifested itself in terms of enabling new applications that have both societal and scientific significance. As a consequence, they have motivated new commercial and business enterprises as well as a considerable body of research.

Because of the many challenges they pose, protocols for self-organizing networks have been the topic of extensive research and an impressive body of work addressing solutions to these challenges exist. However, while theoretical solutions that have been proposed to date are countless, one can quite easily index the few fully-functional implementations. Some of the reasons for this gap between theory and practice include:

- *Complexity.* Protocols are becoming more and more complex, which makes them more difficult to implement.
- *Implementation is difficult.* Protocol implementation typically requires a number of specific skills such as kernel programming, implementing device drivers, operating system knowledge, and many more.
- *Specific solutions.* Implementations are very often specific to a particular solution. Thus, code reuse is difficult, increasing the efforts necessary to develop new approaches.
- *Time consuming.* Fully implementing a new protocol is quite time consuming and sometimes not regarded as fundamental research.

In the area of self-organizing networks, wireless mesh networks (WMN) are particularly interesting because of their

This work has been partially supported by the European Commission project WIP under contract 27402 and by the RNRT project Airmet under contract 01205.

practical interest [1]. Designing protocols for wireless mesh networks poses countless technical challenges due to a variety of factors such as the unpredictable characteristics of the wireless medium and the mobility of clients. This implies that testing and evaluating such protocols under real operating conditions is crucial to ensure adequate functionality and performance, as witnessed by the numerous testbeds deployed by the research community [2]–[6].

The goal we set out to accomplish is to help bridging the gap between theoretical solutions and practical implementations in the context of wireless networks. To this end, we advocate including *rapid prototyping* as part of the design process of wireless protocols, as shown in Fig. 1. Traditionally, prototyping in wireless networks (but also in other contexts) consists of developing a beta version of the final implementation, which essentially corresponds to an approaching step before a public release. Therefore, testing a protocol under real conditions often happens at the end of the development cycle or even after it is over (cf., Fig. 1a). We advocate that prototyping should become an integral part of the protocol design process, as protocol designers can quickly prototype their solution early in the design cycle (cf., Fig. 1b). Rapid prototyping relies on high-level tools to realize a functional, although not necessarily optimized, version of a system. This allows conducting correctness verification, functionality tests under real-world conditions, refinements, and tuning according to the expected behavior of the system; and all this with little development effort. Contrary to the final implementation, rapid prototyping does not require to deal with practical considerations such as opening sockets or sending system calls to the kernel. Prototyping relies on implemented building blocks that make implementation easy and intuitive. Prototyping is done early enough in the design process so that the feedback resulting from testing the system under real conditions can be incorporated into the design.

In this paper, we introduce EPEW, a software development environment that makes prototyping of wireless mesh network protocols on top of IEEE 802.11 considerably easier and faster. EPEW relies on the same principles of Prawn, a prototyping software for flat wireless networks [7]. However, EPEW brings significant improvements over Prawn as it is designed to handle the multi-tiered architecture of wireless mesh networks (as explained in Section III).

EPEW runs as a background process that *proactively* performs tasks such as neighbor discovery and link quality assessment. This feature allows EPEW to provide accurate

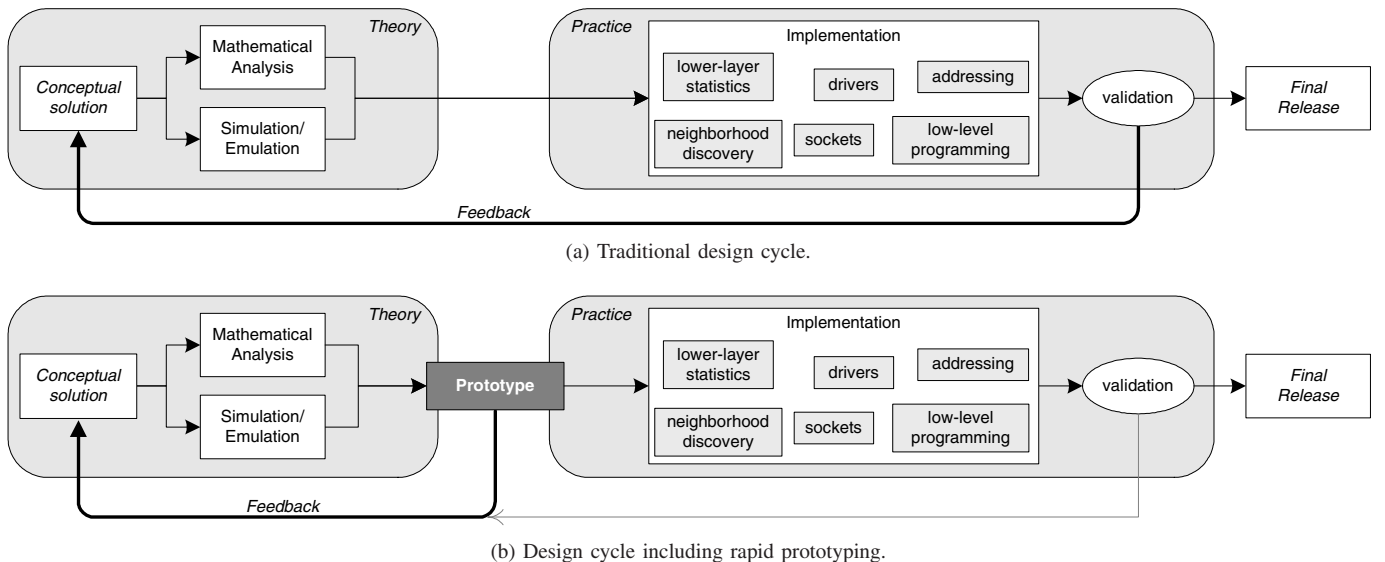


Fig. 1: Rapid Prototyping in the design cycle of protocols and systems for wireless networks.

and up-to-date feedback from the wireless interface. Some previous work share some of the goals of Prawn, but, as further explained in Section V, are either specific to certain types of networks [8], [9] or provide only a passive access to lower level functionalities [10], [11]. To our knowledge, none of them are able to provide a proactive two-way interface with the wireless device while offering high-level communication primitives, all grouped under a single framework.

As a summary, to help prototyping protocols for wireless mesh networks, EPEW provides the following features:

- 1) EPEW fully supports multiple wireless interfaces instead of one. In its basic functioning, it manages one interface in ad hoc mode (for the communication through the backbone), and a second one in access point (AP) mode (to provide access to clients).
- 2) EPEW includes three modes of operation: (i) *ad hoc mode*, to use only one wireless interface in ad hoc mode, (ii) *mesh mode*, to use two wireless interfaces as a mesh router, and (iii) *client mode*, to use only one interface in infrastructure mode as a client.
- 3) Depending on the chosen mode, EPEW is able to configure automatically the wireless interface(s), using appropriate IP configurations and IEEE 802.11 parameters (e.g., interface mode, ESSID, frequency).
- 4) EPEW's functionalities are suited to Wireless Mesh Networks. For instance, in *client mode*, users can retrieve the list of available access points and associate to one of them. In *mesh mode*, the packets to send are automatically routed to the right interface and users can retrieve the list of clients associated to the access point interface.

The remainder of this paper is organized as follows. Next section provides an overview of Prawn. EPEW is then described in Section III and some examples of use are presented in Section IV. We put our work in perspective by reviewing related work in the last section. Finally, we present our con-

cluding remarks and directions for future work in Section VI.

## II. RAPID PROTOTYPING

In order to be consistent with the literature, we first provide a brief overview of the Prawn architecture [7], which defines the basis over which EPEW has been designed. The target is to prototype protocols and services at the network layer and above. Simplicity was a major goal we had in mind when designing Prawn; we wanted to ensure that learning how to use Prawn would be as intuitive and immediate as possible requiring only basic programming expertise. For example, sending a packet at a given transmit power level is performed by a single primitive and takes one line of code. Our focus was to provide: (1) a concise, yet complete set of functions to realize high-level protocols and (2) a simple, easy-to-use and intuitive interface to provide access to the functionalities.

The architecture of our prototyping environment is based on two main components: (i) the Library (cf., Section II-A), which provides high-level primitives to send and receive messages, retrieve information from the network, etc; and (ii) the Engine (cf., Section II-B), which implements the primitives provided by the Library.

### A. The Library

The Library provides a set of high-level communication-oriented functions. They hide from protocol designers lower-level features such as addressing, communication set-up, etc. Their syntax is simple and intuitive, and the set of primitives addresses basic functions required when prototyping a high-level communication protocol. Nevertheless, this list can be easily further extended allowing new primitives to be implemented and integrated.

- `Prawn_Info()` returns information on the configuration of the local Engine. Basically, it consists of the list of settings chosen when launching the daemon (cf., Table I).

Some examples are the node's ID, interface port number, and beacon period.

- `Prawn_Neighbors()` returns the list of the node's one-hop and two-hop neighbors as well as statistics concerning the quality of the respective links. In Section II-B, a thorough explanation of the information returned by the Engine will be given.
- `Prawn_Send(Message, ID, TX_Pwr)` sends `Message` to node `ID`; the optional argument `TX_Pwr` can be used to explicitly set the transmit power to be used during the transmission. `Message` can be a string, a number, a data structure, or any other data or control message, depending on the prototyped protocol.
- `Prawn_Send_Broadcast(Message, TX_Pwr)` sends a broadcast message containing `Message`; in a similar way to `Send()`, the optional argument `TX_Pwr` allows to set the transmit power.
- `Prawn_Receive()` checks if a message has been received; if so, the message is returned. This primitive is non-blocking: if no message has been received, it just returns zero.
- `Prawn_Receive_Block()` checks if a message has been received; if so, the message is returned. This primitive is blocking and awaits for a message to be received before returning it.

### B. The Engine

The Engine is event-driven, i.e., its main process remains asleep waiting for an event to occur. An event can be triggered by a request from the Library (coming through the loop-back interface) or by a message received on a wireless interface. Meanwhile, local neighborhood discovery is performed through beacon and feedback packets.

To build and maintain the list of neighbors, the Engine broadcasts 24-byte beacons periodically. The beacon period is configurable depending on the requirements of the prototype under development. By default, the Engine is configured to test connectivity under different power levels (useful for instance to prototype topology control algorithms based on power control). The Engine applies a round-robin policy to continuously change the transmit power. A beacon is first broadcast with the lowest power value. The transmit power level is successively increased for each beacon, up to the maximum transmit power. We call this sequence of beacons a *cycle*. The different values of the transmit power are either obtained from the interface or set by the user. This cycle is then repeated at every beacon period. This way, the time elapsed between two beacons sent with the same transmit power is equal to the beacon period. Upon the reception of a beacon (or sequence of beacons if different transmit powers are used), various statistics can be derived.

Nodes reply to beacons using 16-byte feedback packets. Feedback packets summarize neighborhood- and link quality information as perceived by the receiver of the beacons. This feature allows verifying the bidirectionality of links. Feedback packets are sent to every neighbor after a complete cycle.<sup>1</sup> The

<sup>1</sup>Note that if the power control feature is disabled, then the cycle is unitary.

Engine keeps sending feedback packets also in the case where a neighbor is considered lost (a unidirectional link may still exist between the two nodes).

Two other key functions performed by the Engine are transmission and reception of data (triggered by the `Prawn_Send()` and `Prawn_Receive()` primitives, respectively). The Engine is in charge of the communication set up, namely opening sockets, converting the receiver identifier to a valid IP address, encapsulating/decapsulating packets, and adjusting the transmit power before transmission. Data packets are sent using UDP to the corresponding IP address. On the receiver side, the Engine listens on an open socket for any incoming packets. Packets are then decapsulated and sent to the prototype, which retrieves them by using the `Prawn_Receive()` primitive.

## III. PROTOTYPING WITH EPEW

Wireless mesh networks have specific needs that could not be satisfied by Prawn. In a WMN, nodes are either mesh routers or clients. The EPEW Engine now supports both types of nodes, and the EPEW Library includes specific primitives for WMNs, as described in the following.

### A. How to Use EPEW

EPEW is distributed under GPL license. The current implementation runs on Linux atop IP for backward compatibility with the global Internet. As it was already the case for Prawn, running EPEW also requires only few basic steps. EPEW features an auto configuration option that generates a random IP address in a default subnetwork, and tries to configure the wireless interfaces with default settings (e.g., ad hoc mode, channel, ESSID), that can be changed in the configuration file (`epew.cfg`).

Using EPEW itself only requires two operations (as described below), namely executing the EPEW Engine and including the EPEW Library in the prototype code.

**1: Starting the EPEW Engine.** EPEW is supposed to run in daemon mode (as a background process), but can run in console mode or graphic mode for debugging purposes. As stated before, EPEW provides a number of options that can be set/configured at the execution of the EPEW Engine. They are listed in Table I. Other options are tunable in the `epew.cfg` configuration file.

**2: Using the EPEW Library.** The EPEW Library (described in detail in Section II-A) is composed of a set of primitives that are linked to the prototype through standard include files. Currently, prototypes can be developed either in C or Perl. For C development, the file `epew.h` should be included in the header of the prototype code. Similarly, the file `epew.pl` is to be included for prototypes developed in Perl.

### B. The EPEW Engine

While Prawn only supports one interface configured in ad hoc mode, mesh routers use two interfaces and clients use an interface in infrastructure mode. EPEW supports two interfaces for mesh routers, as well as the infrastructure mode for both clients and mesh routers' access point interface.

TABLE I: EPEW's command line options.

Option	Parameter	Default
-a	auto-configuration	-
-b period	beacon period in ms	10000
-c port	client port	3020
-C	access point client mode	-
-d	daemon mode	-
-g	graphic user interface	-
-h	help	-
-i I	uses wireless interface I	ath0
-M	mesh router mode	-
-n	no power control features	-
-N name	node ID	hostname
-p port	neighbor port	3010
-v	verbose mode	-
-vv	more verbose	-
-V	version	-
-W window	window size for PER	5

The EPEW Engine is based on the Prawn Engine, but can now be launched in three different modes, depending on the host role:

- Normal mode. When no special mode is specified in the command line, EPEW is launched as a regular node in an ad hoc network. Only one wireless interface is used, and, if enabled, the auto configuration option sets it in ad hoc mode.
- Mesh router mode. This mode enables 2 wireless interfaces, the first one to communicate with other ad hoc routers, and the second one to communicate with access point clients. If enabled, the auto-configuration option sets the first one in ad hoc mode, and the second one in master mode (to be used as an access point). The IP address of the router interface is set randomly as usual, whereas the one of the access point interface is set to a fixed default address in a different default subnetwork.
- Client mode. This mode only uses one wireless interface, configured in infrastructure mode by the auto-configuration option. The IP address is set randomly in the same default subnetwork as the access point.

To set the node's mode, EPEW includes two new options (-M and -C), as referred in Table I. Of course, only one option can be chosen, or none of them to use the node as a regular ad hoc node.

For mesh routers, beacon packets and feedback packets are sent through both interfaces. The engine opens two different sockets and listens to received packets from both sockets. Two neighbor lists are updated when receiving beacon and feedback packets, as shown in figure 2. The first one concerns the other neighboring mesh routers, and the other one is the list of clients associated to the access point interface.

### C. The EPEW Library

To provide new functionalities needed by algorithms for wireless mesh networks, EPEW provides an extended list of primitives as follows:

```

=====Neighbor List @7270:=====
2BE0 Active 06:60:B3:AC:2B:E0 Beacon period : 1000 nodeA
Weakest beacon received by this neighbor : 31 mW
-----
100mW Active @7270 R 0 S 783 B 42 [0.063095734 nW (-72 dBm)] 5/5

2hop-Neighbors : nodeC (lost) nodeB (100 mW, -34 dBm)
=====
A242 Active 06:60:B3:AC:A2:42 Beacon period : 1000 nodeB
Weakest beacon received by this neighbor : 31 mW
-----
100mW Active @7269 R 0 S 799 B 51 [316.227766017 nW (-35 dBm)] 5/5

2hop-Neighbors : nodeC (lost) nodeA (100 mW, -72 dBm)
=====
=====Access Point Neighbor List @7270:=====
2C2C Active 06:60:B3:AC:2C:2C Beacon period : 1000 nodeD
Weakest beacon received by this neighbor : 100 mW
-----
100mW Active @7270 R 0 S 311 B 40 [398.107170553 nW (-34 dBm)] 5/5

2hop-Neighbors : nodeE (100 mW, -3 dBm)
=====
2B55 Active 06:60:B3:AC:2B:55 Beacon period : 1000 nodeE
Weakest beacon received by this neighbor : 100 mW
-----
100mW Active @7270 R 0 S 402 B 41 [794.328234724 nW (-31 dBm)] 5/5

2hop-Neighbors : nodeD (100 mW, -4 dBm)
=====
>_

```

Fig. 2: Snapshot of EPEW running in console mode, and showing the list of neighbors in both interfaces of a mesh router.

- EPEW\_Neighbors(): In mesh router mode, the list returned concerns the ad hoc interface and provides the list of mesh routers in range. In client mode, only returns information about the current associated access point.
- EPEW\_Clients(): This new primitive is only for mesh router mode. It provides the list of clients associated to the access point interface, along with information on link characteristics.
- EPEW\_Send(Message, ID, TX\_Pwr): As in the flat approach, it sends Message to node ID; the optional argument TX\_Pwr can be used to explicitly set the transmit power to be used during the transmission. In mesh router mode, the engine looks for the destination in both lists (neighbors list and client list), and send the message to the appropriate interface. In client mode, the node can only send messages to the access point (which is the unique neighbor).
- EPEW\_Send\_Broadcast\_AP(Message, TX\_Pwr): To be used only in mesh router mode, this primitive sends a broadcast message containing Message to all the clients, through the access point interface.
- EPEW\_Scan(): Only available in client mode, this primitive requests a scanning of in range access points (with the same ESSID) and returns a list of detected access points, along with their respective MAC address, signal level, and identifier (which is a simple number to avoid handling the MAC address). This list is sorted according to the signal level.
- EPEW\_Associate(AP): Also exclusively available in client mode, this last primitive associates to the access point AP. AP is the requested access point identifier as in the list returned by EPEW\_Scan().

Other primitives like Prawn\_Receive() remain unchanged and can be used with EPEW\_Receive() for example.

## IV. CASE STUDIES

EPEW is intended to be a tool for prototyping a wide range of communication algorithms for wireless mesh networks, including the algorithm involving routers (in the ad hoc network) and also handling client association (hand over, mobility, etc). In this section, we illustrate the use of Prawn through very simple examples, highlighting its range of applicability and ease of use.

### A. “Hello World!”

First, we describe how to implement a simple “hello world” prototype using EPEW’s Perl library. In this example we send a message from Bob to Alice. Note that Bob and Alice must be connected through a wireless link, but can be either two mesh routers (ad hoc link), or a client and a mesh router (infrastructure link).

**Step 1.** Launch Prawn with “prawn -d -N Bob” in the first machine and “prawn -d -N Alice” in the second machine.

**Step 2.** Get the first machine ready to receive messages by executing the following Perl script:

```
require "epew.pl";

$message=EPEW_Receive_Block();
print "Received $message->{Data} from $message->{Sender}";
```

**Step 3.** On the other machine launch the following Perl script:

```
require "epew.pl";
EPEW_Send("Hello World!","Bob");
```

The result is as expected: Alice sends a “Hello World” message to Bob, and Bob prints “Received: Hello World! from Alice” on the screen. This simple example aims at showing the level of abstraction provided by Prawn, where low-level system knowledge (e.g., sockets, addressing) is not required.

### B. Basic chat application

The following scripts prototype a basic chat application (alternate conversation) in a multi-hop path. Two client nodes, *node1* and *node2* communicate through the same mesh router (access point) *node3*. The mesh router runs the following EPEW’s script:

```
require "epew.pl";

while(1){
  $message=<>;
  EPEW_Send($message,"node3");
  $message=EPEW_Receive_Block();
  print "Received: $message->{Data}";
}
```

As for *node1* (which initiates the conversation), it runs the following script:

```
require "epew.pl";

while(1){
  $message=EPEW_Receive_Block();
  if ($message->{Sender} eq "node1"){
    EPEW_Send($message->{Data}, "node2");
  }
  else{
    EPEW_Send($message->{Data}, "node1");
  }
}
```

### C. Handling Access Points

Associating to access points is made very easy and intuitive with EPEW, with the following script:

```
require "epew.pl";

AP_list=EPEW_Scan();
EPEW_Associate(1);
print "Associating to AP 1 \n";
print "(Signal level: $AP_list[1]->{level} dBm \n";
```

The node requests the list of available access points, and chooses the access point with the best signal level (ranked first) to associate with. The signal level of the chosen access point is then displayed.

## V. RELATED WORK

Simulations are perhaps the most widely used methodology for evaluating network protocols. They allow designers to evaluate the system at hand under a wide range of parameters like different mobility models and node heterogeneity, but only under synthetic channel models. Simulation has the advantage of allowing the exploration of the design space by enabling designers to vary individual protocol parameters (e.g., timers) and combinations thereof. Finally, they are instrumental for scalability analysis and they offer reproducibility. Examples of well-known simulation platforms include NS-2 [12], GloMoSim [13], and OPNET [14].

Emulation tries to subject the system under consideration to real inputs and/or outputs. Environments like EMPOWER [15] or Seawind [16] emulate the wireless medium by introducing packet error rates and delays. Other emulators like m-ORBIT [17] also emulate node mobility by space switching over a testbed of fixed nodes. A key advantage of emulation in the context of wireless/mobile networks is to facilitate testing by avoiding, for example, geographic and mobility constraints required for deployment.

More recently, a number of projects have pioneered the field of wireless protocol evaluation under real conditions. They include testbeds such as Orbit [18], and Roofnet [2] as well as tools that support protocol implementation like Click [19] and XORP [20]). As previously pointed out, such tools and EPEW have different goals, address different phases of the design process, and are therefore complementary. While tools like Click and XORP target the implementation at the final stages of protocol design, EPEW focuses on prototyping a research proposal at the very early stages of the design process. Therefore, through EPEW, protocol designers can very quickly and easily generate a fully functional, although non-optimized, implementation for live testing in real scenarios.

In the context of wireless sensor networks (WSNs), Polastre *et al.* [8] propose SP (Sensornet Protocol), a unifying link abstraction layer. SP provides an interface to a wide range of data-link and physical layer technologies. EPEW and SP roughly share the same functional principles, e.g., data transmission, data reception, neighbor management with link quality, etc. However, they have different goals. First, SP only manages the neighbor table; it neither performs neighbor discovery nor provides link assessment. Second, SP is designed for WSNs whereas EPEW is for IEEE 802.11

networks, and particularly for WMNs. Finally, while SP aims at optimizing the communication and unifying different link layers in WSNs, EPEW aims at facilitating and simplifying prototype implementation.

## VI. CONCLUSION

In this paper we presented EPEW, the extension to Prawn for prototyping high-level WMN protocols and applications. EPEW's main goal is to facilitate the prototyping of WMN protocols so that prototyping becomes an integral part of the design process of wireless systems.

EPEW is not an alternative to simulation or any other evaluation method. Instead, it stands as a complementary approach that goes beyond simulation by taking into account real-world properties. EPEW surfs the wave of recent research efforts toward making implementation easier (e.g., Click and XORP), but as a preliminary phase in this process.

Experiments where EPEW can be useful for WMN testbeds include: testing hand over algorithms and mobility management, evaluating existing protocols for wired networks in the wireless context, implementing new routing protocols, testing overlay approaches in wireless multi-hop networks, evaluating distributed security algorithms, testing new naming mechanisms over IP, testing incentive mechanisms for communities, implementing localization algorithms, measuring wireless connectivity in both indoor and outdoor scenarios, evaluating peer-to-peer algorithms, testing opportunistic forwarding mechanisms.

Our work is aimed at providing a starting point for an improved design methodology as prototyping allows easy and accurate evaluation of protocols and services for wireless mesh networks under real conditions.

## REFERENCES

- [1] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," *Computer Networks*, vol. 47, no. 4, pp. 445–487, Jan. 2005.
- [2] D. Aguayo, J. Bicket, S. Biswas, and R. Morris. MIT roofnet. [Online]. Available: <http://pdos.csail.mit.edu/roofnet/doku.php>
- [3] H. R. Lundgren, K. N. Ramachandran, E. M. Belding-Royer, K. C. Almeroth, M. Benny, A. Hewatt, A. Touma, and A. P. Jardosh., "Experiences from building and using the ucsb meshnet testbed," *IEEE Wireless Communications Magazine: A Special Issue on Mesh Networks*, 2006.
- [4] Y. Takahashi, Y. Owada, H. Okada, and K. Mase, "A wireless mesh network testbed in rural mountain areas," in *WINTECH*, Montréal, Canada, 2007, pp. 91–92.
- [5] H. Song, B. C. Kim, J. Y. Lee, and H. S. Lee, "Ieee 802.11-based wireless mesh network testbed," in *16th IST Mobile and Wireless Communications Summit*, Budapest, Hungary, 2007.
- [6] K. chan Lan, Z. Wang, M. Hassan, T. Moors, R. Berriman, L. Libman, M. Ott, B. Landfeldt, and Z. Zaidi, "Experiences in deploying a wireless mesh network testbed for traffic control," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 5, pp. 17–28, 2007.
- [7] F. Ben Abdesslem, L. Iannone, M. D. de Amorim, K. Obraczka, I. Solis, and S. Fdida, "A prototyping environment for wireless multihop networks," in *AINTEC*, Phuket, Thailand, 2007, pp. 33–47.
- [8] J. Polastre, J. Hui, P. Levis, J. Zhao, D. E. Culler, S. Shenker, and I. Stoica, "A unifying link abstraction for wireless sensor networks." in *Proceedings of ACM Sensys*, 2005, pp. 76–89.
- [9] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, "Emstar: A software environment for developing and deploying wireless sensor networks." in *Proceedings of USENIX Annual Technical Conference*, 2004, pp. 283–296.
- [10] C. M. T. Calafate and P. Manzoni, "A multi-platform programming interface for protocol development." in *Proceedings of the Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, Genova, Italy, Feb. 2003, pp. 243–249.
- [11] M. Youssef. MAPI: An API for wireless cards under linux. [Online]. Available: <http://www.cs.umd.edu/moustafa/>
- [12] The Network Simulator NS-2. [Online]. Available: <http://www.isi.edu/nsnam/ns/>
- [13] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a library for parallel simulation of large-scale wireless networks," in *Proceedings of Workshop on Parallel and Distributed Simulation*, Banff, Canada, May 1998.
- [14] OPNET Modeler. [Online]. Available: <http://www.opnet.com/products/modeler/>
- [15] P. Zheng and L. Ni, "EMPOWER: A network emulator for wireless and wireline networks," in *Proceedings of IEEE Infocom*, San Francisco, CA, Apr. 2003.
- [16] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. O. Alanko, and K. E. E. Raatikainen, "Seawind: a wireless network emulator," in *GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems*, Aachen, Germany, Sep. 2001.
- [17] K. Ramachandran, S. Kaul, S. Mathur, M. Gruteser, and I. Seskar, "Towards large-scale mobile network emulation through spatial switching on a wireless grid," in *Proceedings of ACM Sigcomm*, Philadelphia, PA, Aug. 2005.
- [18] D. Raychaudhuri, M. Ott, and I. Seskar, "Orbit radio grid tested for evaluation of next-generation wireless network protocols." in *Proceedings of Tridentcom*, 2005, pp. 308–309.
- [19] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.
- [20] M. Handley, O. Hodson, and E. Kohler, "XORP: an open platform for network research," *Computer Communications Review*, vol. 33, no. 1, pp. 53–57, 2003.