

# An Abstraction Layer for Neighborhood Discovery and Cross-Layer Metrics

Fehmi Ben Abdesslem<sup>1</sup>, Luigi Iannone<sup>1</sup>, Marcelo Dias de Amorim<sup>1</sup>,  
Katia Obraczka<sup>2</sup>, Ignacio Solis<sup>3</sup>, and Serge Fdida<sup>1</sup>

<sup>1</sup> Laboratoire LIP6/CNRS  
Université Pierre et Marie Curie – Paris 6  
Paris, France  
{fehmi, iannone, amorim, sf}@rp.lip6.fr

<sup>2</sup> Computer Engineering Dept.  
University of California, Santa Cruz  
Santa Cruz, CA, USA  
katia@cse.ucsc.edu

<sup>3</sup> PARC  
Palo Alto Research Center  
Palo Alto, CA, USA  
isolis@parc.com

**Abstract**—Topology-awareness is an important feature of communication algorithms and protocols for wireless multi-hop networks. In its basic form, topology-awareness requires nodes to know their immediate neighbors and, if possible, the quality of the links between them. In general, neighborhood discovery is done by routing protocols; the problem with such an approach is that fundamental changes in node characteristics require revisiting the routing protocol, especially if this latter relies on cross-layer metrics. In this paper, we propose an abstraction layer that performs neighborhood discovery and link quality assessment. In this way, neighborhood discovery is decoupled from the upper layers. Our abstraction layer stands as an active building block for the implementation of routing protocols or any other IEEE 802.11 communication algorithm for wireless multi-hop networks. It provides an updated list of neighbors, along with several statistics about link quality that can be used, for instance, to compute cross-layer metrics.

## I. INTRODUCTION

Wireless networks introduce new constraints when compared to wired networks. In the wired case, the characteristics of the physical medium is in general well known; nodes listening to a transmission can be easily determined, and applications and protocols are often based on a known topology. Nevertheless, the advent of unlicensed spectrum, cheap wireless interfaces, and the inherent convenience of untethered computing have made wireless-based networks ubiquitous in the current scenario of networking.

In their basic definition, wireless networks, and more specifically wireless multi-hop networks, introduce the constraint that the topology is potentially dynamic and unknown a priori. These networks are also expected to be self-organizing [1], [2], [3]. In order to provide self-organizing functionalities to nodes, the fundamental step is to make nodes be capable of sensing their environment. The environment is basically characterized by the set of neighboring nodes; however, this is not enough in the wireless case because of the highly dynamic nature of the channel. Hence, nodes also need to be aware of the conditions of the links toward each of the neighbors.

Implementing extended neighborhood discovery is difficult, requiring time and programming skills. Thus, in order to

simplify the development process, there is a need of a solution that provides ready-to-use information about the neighborhood and the link quality. In this domain, a number of proposals have been issued to provide a better understanding of the wireless medium. Nevertheless, they are in general either an indicator of the neighbors in the vicinity of a node or a simple definition of a new metric to evaluate the wireless medium (cf., Section II). To our knowledge, none of them provide an integrated solution that allows to retrieve fundamental metric values such as reception quality, link attenuation, power levels per node, and others.

In this paper, we propose an abstraction layer that implements active neighborhood discovery and link quality assessment. This abstraction layer stands as a building block implemented in C language to simplify the development of communication algorithms for wireless networks. The mechanism consists in sending broadcast beacons with different transmit power and collecting statistics about the quality of the links. Another value-add of our abstraction layer is that it allows routing protocols make abstraction of lower-layer (independent) tasks and focus on the objectives it has been designed for.

We show the usefulness of our system with both quantitative and qualitative arguments. In particular, we show the adaptive features of our tool through its tunable parameters such as the beaconing period and the transmit power levels. Our results show that this is fundamental to capture the channel characteristics at the accuracy level required by the user.

The remainder of the paper is organized as follows. In Section II, we present the related work and the main differences that make the originality of our solution. The two building blocks of our tool, neighborhood discovery and cross-layer metrics are detailed in Section III. In Section IV, we present the implementation details of our tool, followed by a number of usage case studies. Finally, Section V discusses the main contributions of this work and identifies future work.

## II. RELATED WORK

Most of the works proposing abstraction layers for the implementation of wireless communication algorithms fall in the area of wireless sensor networks (WSNs). Polastre

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

et al. propose SP (Sensornet Protocol) [4], a unifying link abstraction layer. SP runs on top of TinyOS [5] and provides an interface to a wide range of data link and physical technologies. Our abstraction layer and SP roughly share the same functional principles: neighborhood management and link quality information. However, they have quite different goals. First, SP only manages the neighborhood table; it neither performs neighborhood discovery nor provides link assessment. Second, SP is designed for WSNs, whereas we focus on general IEEE 802.11 networks. Finally, while SP aims at optimizing the communication and unifying different link layers in WSNs, whereas our abstraction layer aims at simplifying the implementation of a prototype.

EmStar [6] is another development environment for WSNs, running on the Intel Stargate platform [7]. It is similar in essence to our abstraction layer since it provides a set of primitives that upper layers can use along with neighborhood monitoring. However, Emstar does not perform active neighborhood discovery nor link quality assessment.

Only a few works have been done in the context of general IEEE 802.11 based networks. The PICA API [8] roughly draws similar objectives as ours. This API allows specific operations such as changing the routing table, opening a socket, writing in a file, and managing threads and memory, to cite a few. Nevertheless, it neither implements explicitly neighborhood discovery nor link quality assessment, and does not provide cross-layer information. Furthermore, PICA does not focus on wireless communication and does not take into account this particular context.

MAPI [9] is an API especially developed for wireless networks, and based on the Wireless Tools [10]. It provides a set of simple primitives to obtain information from the wireless device. Contrary to MAPI, our abstraction layer does not only provide information from the wireless device through a simple API, but also an active daemon implementing neighborhood discovery with link assessment.

### III. AN ACTIVE ABSTRACTION LAYER

The abstraction layer we propose in this paper is composed of two main components: neighborhood discovery and collection of cross-layer metrics. We explain these two components in the following.

#### A. Neighborhood discovery

Our abstraction layer performs two main tasks. First, it responds to the requests of the user by keeping an updated list of neighbors and corresponding link quality information. Second, through an active round-robin mechanism, it performs neighborhood monitoring that maintains up-to-date neighborhood information.

To build and maintain the list of neighbors, each node running the abstraction layer periodically broadcasts 24-byte beacons, varying the transmit power for each beacon. The number of transmit power levels and their values are customizable, depending on the power control features provided by the wireless interface being used. A round-robin policy is used to continuously change the transmit power. Beacons are

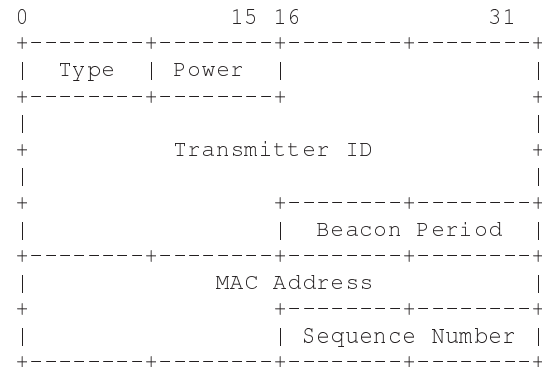


Fig. 1: The beacon packet format

first broadcasted with the lowest power value; the power value level is then increased at predefined levels up to the maximum transmit power (obtained from the interface or set by the user). This cycle is repeated at every beaconing period.

The beaconing packet format, illustrated in Fig. 1, includes the following fields:

- **Type:** Type of the packet (in this case, the type is “beacon packet”, identified by the value 1).
- **Power:** Transmit power at which the packet is being sent (useful for the receiver to infer the quality of the link, as we will see in Section III-B).
- **Transmitter ID:** Sender identifier. This field allows the user to set an user-friendly identifier to the node.
- **Beacon Period:** Time period between two beacons transmitted with the same power level (set by the user and also used by the receiver to infer some metrics).
- **MAC Address:** MAC address of the sender.
- **Sequence Number:** Sequence number of the beacon.

Configuring the abstraction layer is important to achieve an adequate balance between performance and overhead. For example, sending beacons too frequently would incur, on the one hand, high overhead. On the other hand, restraining the number of beacons is likely to result in out-of-date measures. For these reasons, the beaconing period is one of the customizable parameters of the abstraction layer and its value is carried in one of the beacon’s fields. A beacon sent at a given transmit power is considered as lost when not received before the beacon period (included in the previous received beacons) times out. By default, a neighbor is removed from a node’s neighborhood table when all the beacons transmitted by this neighbor with every transmit power level have been lost. The number of beacon periods to wait before a beacon is assumed to be lost is also customizable by the user.

All nodes within the communication range of the transmitter will include this latter in their neighborhood table. Since all nodes apply the same procedure, full neighborhood knowledge is incrementally obtained even if beacon packets are subject to losses (thanks to the periodic algorithm). In the following, we will show how nodes obtain quality information from the underlying medium.

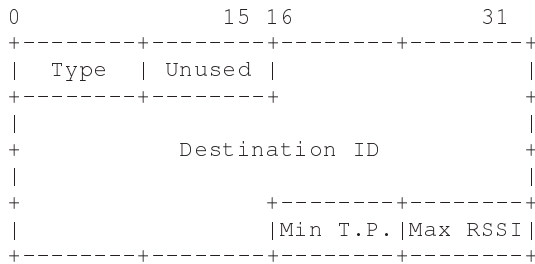


Fig. 2: The feedback packet format.

### B. Collecting cross-layer metrics

Upon the reception of a beacon (and ideally of a cycle of beacons), various statistics can be derived and can be used as cross-layer metrics. For instance, a node A can determine, at a given point in time, the minimum transmit power that B should use to send messages to A. This value corresponds to the lowest transmit power among all the received beacons from B. Of course, the minimum transmit power may change over time, and will be updated as nodes frequently beacon their neighbors. Another example is the received signal strength of the last beacon received by a neighbor.

To collect more statistics and better characterize the link quality, nodes reply to beacons using 16-byte packets (or feedback packets) whose fields are described below and depicted in Fig. 2. Feedback packets collect information on the neighborhood and link quality as perceived by the destination of the original beacon. This feature also allows verifying if links are bidirectional. Feedback packets are sent to every neighbor after a complete cycle of beacons, even to neighbors that have not replied to these beacons. The abstraction layer keeps sending feedback packets also in the case where a neighbor is considered lost (a unidirectional link may exist between the two nodes)<sup>1</sup>. Feedback packets contain the following fields:

- **Type:** Type of the packet. Here, the type is “feedback packet”, identified by the value 3).
- **Destination ID:** Neighbor Identifier.
- **Minimum Transmit Power Received:** Lowest transmit power level among the beacons received from that neighbor.
- **Maximum Power Strength Received:** Maximum RSSI (Received Signal Strength Indicator) measured among the received beacons from that neighbor.

Feedback packets notify each beaconing node what is the transmit power of the weakest beacon received from that neighbor. The weakest beacon received by a node allows it to roughly characterize the quality of the corresponding link. For example, if a node receives only beacons transmitted at 20 mW and above, it will send this value in its feedback packet. Additionally, feedback packets include the maximum received signal strength (in dBm) measured at beacon reception.

Although they are destined to a single neighbor, feedback packets are broadcast and thus overheard by all one-hop neighbors. This way, two-hop neighborhood and link quality

<sup>1</sup>This option can be switched off, to avoid wasting bandwidth with ghost nodes.

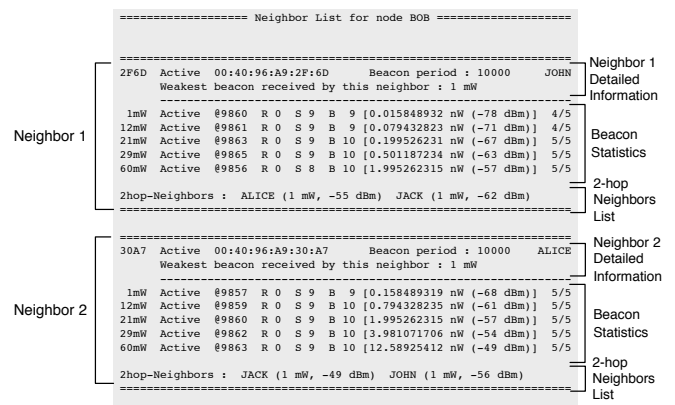


Fig. 3: Example of the information provided by the abstraction layer for a node whose ID is “BOB”.

information is disseminated. Below, we explain how neighborhood information gathered by the abstraction layer is provided to the user through the console mode. Note that the console mode is to be used for monitoring or debugging purposes. A routing protocol will interact with the abstraction layer launched in daemon mode, through the loopback interface.

### C. Example

Fig. 3 shows a snapshot of the information returned by the abstraction layer, running in console mode on a node named “Bob”. Basically, Bob has two active neighbors, John and Alice, and one two-hop neighbor, Jack. As we can see from the snapshot, Bob is able to receive all beacons sent by both John and Alice. Nevertheless, in average, the link between Bob and Alice has better quality than the one between Bob and John.

### D. Further features

As previously described, neighborhood information is obtained through beacons and feedback packets. More specifically, broadcast beacons are used to build the list of direct neighbors. This list is established by gathering the transmitter ID of each received beacon. Moreover, data included in beacons and feedback packets inform each node what is the minimum transmit power required to reach a neighbor. Such information is of primary importance in assessing link quality. Another prominent link characteristic is the error rate, which is determined according to the beacon period included in each beacon transmitted. The engine considers a beacon as lost when it is not received within the beacon period indicated by the corresponding neighbor. The size of the receiving window used to compute the error rate is customizable. For instance, in Fig. 3, the error rate for John’s packets transmitted at 12 mW is 1/5, because over the 5 most recent 12 mW beacons transmitted by John, only 4 have been received.

When receiving a beacon, the abstraction layer retrieves and saves the received signal strength. Along with the transmit power of the beacon (which is also included in the beacon), the received signal strength returned by the engine helps evaluate signal attenuation. The difference between the transmitted

**Algorithm 1** The abstraction layer's main loop pseudo-code

---

```

1: Timeout ← time_to_next_regular_event
2: while 1 do
3:   if (Timeout) then
4:     Perform regular operation
5:     Timeout ← time_to_next_regular_event
6:   else if (Client Request) then
7:     Return requested information
8:   else if (Neighbor Message) then
9:     Update neighborhood list and statistics
10:  end if
11: end while

```

---

power level indicated in the beacon and the signal strength measured when the beacon is received can also be used to characterize link quality.

#### IV. IMPLEMENTATION AND EXPERIMENTATION

In this section, we explain the implementation details of our abstraction layer and a number of experiments that showcase the facilities provided by the tool.

The current implementation of our abstraction layer runs on Linux for typical equipments like laptops, mini-PCs, and mesh routers. The abstraction layer is also available for Nokia N770 devices, which are PDA-like devices useful to test communication algorithms and protocols in a mobile topology. The interaction with the physical wireless device relies on the Wireless Tools [10]. This set of tools allows the retrieval of information from most wireless devices as well as setting of low-level parameters. However, to retrieve more information from the wireless device, we plan to use more advanced tools like XIAN [11]. The implementation is completely event-driven, i.e., its main process remains asleep waiting for an event to occur. The main loop is described in pseudo-code in Algorithm 1. The most important events are:

- *Control Event*: Control messages have to be performed on a regular basis, controlled by a timer. A timeout event triggers the transmission of neighborhood discovery control messages (beacons or feedbacks).
- *Client Request*: The Client Request event is an on-demand event triggered by a user call requesting an action from the engine (e.g., retrieving the current neighborhood list).
- *Neighbor Message*: The Neighbor Message event also happens on demand and is triggered when packets from neighbor nodes are received through the wireless interface. These packets could be either beacon packet or feedback packets.

The main issue to consider when implementing our abstraction layer is the interface with the physical layer. Several 802.11 cards are available in the market, with different chipsets embedded. Our implementation has been successful for Atheros-based and Intel-based wireless cards, running with the MadWiFi driver [12] and Ipw2200 driver [13] for Linux. These drivers are implemented by contributors and are constantly updated to support more features. The main optional

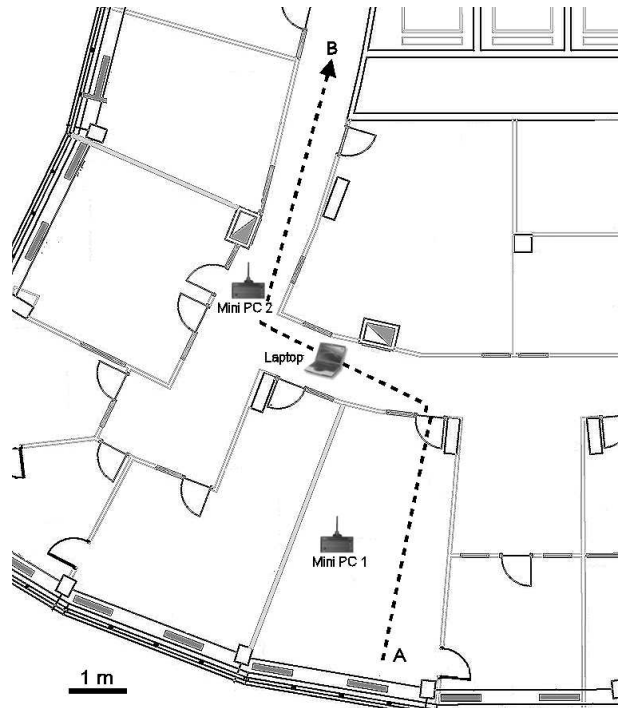


Fig. 4: Plan of our computer science laboratory, representing a laptop moving from A to B.

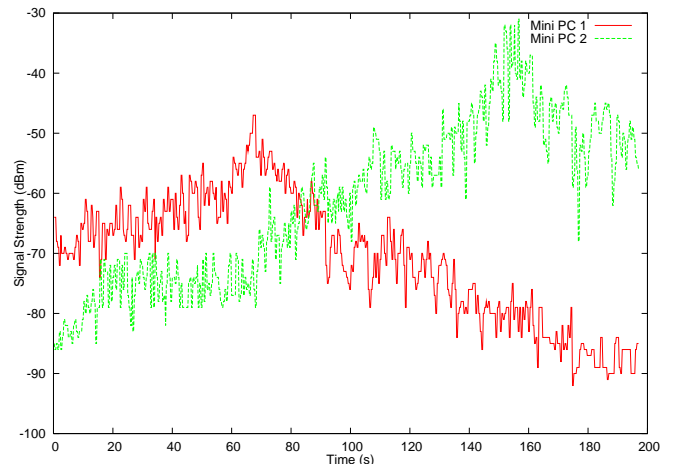


Fig. 5: Received signal strength of Mini PC 1 and Mini PC 2 measured from the laptop.

features that are appreciated when running the abstraction layer are power control capability and the ability to measure the received signal strength. Of course, the abstraction layer can still work without these advanced features. A typical issue for the received signal strength is that both drivers return an average value for the received signal strength, computed exponentially with the last packets received, whereas we need the exact value of the last one.

We propose a practical example of how the abstraction layer can dynamically keep a node aware of its environment. The scenario is shown in Fig. 4 and corresponds to part of the 7th floor of the LIP6 laboratory of the Université Pierre et Marie

```

===== Neighbor List for node LAPTOP=====
3051 Active 00:40:96:A7:30:51 Beacon period : 250 MINIPC1
Weakest beacon received by this neighbor : 60 mW
-----
60mW Active @1223 R 0 S 55 B 56 [19.952623149 nW (-47 dBm)] 5/5
2hop-Neighbors : MINIPC2 (60 mW, -85 dBm)
=====
30A7 Active 00:40:96:A9:30:A7 Beacon period : 250 MINIPC2
Weakest beacon received by this neighbor : 60 mW
-----
60mW Active @1223 R 0 S 64 B 56 [0.012589254 nW (-79 dBm)] 3/5
2hop-Neighbors : MINIPC1 (60 mW, -82 dBm)
=====

```

Fig. 6: Snapshot of the abstraction layer at time 65 seconds

```

===== Neighbor List for node LAPTOP=====
3051 Active 00:40:96:A7:30:51 Beacon period : 250 MINIPC1
Weakest beacon received by this neighbor : 60 mW
-----
60mW Active @1245 R 0 S 133 B 140 [1.995262315 nW (-57 dBm)] 5/5
2hop-Neighbors : MINIPC2 (60 mW, -84 dBm)
=====
30A7 Active 00:40:96:A9:30:A7 Beacon period : 250 MINIPC2
Weakest beacon received by this neighbor : 60 mW
-----
60mW Active @1245 R 0 S 152 B 136 [1.584893192 nW (-58 dBm)] 5/5
2hop-Neighbors : MINIPC1 (60 mW, -82 dBm)
=====

```

Fig. 7: Snapshot of the abstraction layer at time 87 seconds

```

===== Neighbor List for node LAPTOP=====
3051 Active 00:40:96:A7:30:51 Beacon period : 250 MINIPC1
Weakest beacon received by this neighbor : 60 mW
-----
60mW Active @1303 R 0 S 305 B 306 [0.015848931 nW (-78 dBm)] 4/5
2hop-Neighbors : MINIPC2 (60 mW, -85 dBm)
=====
30A7 Active 00:40:96:A9:30:A7 Beacon period : 250 MINIPC2
Weakest beacon received by this neighbor : 60 mW
-----
60mW Active @1303 R 0 S 424 B 407 [630.957344480 nW (-32 dBm)] 5/5
2hop-Neighbors : MINIPC1 (60 mW, -81 dBm)
=====

```

Fig. 8: Snapshot of the abstraction layer at time 155 seconds

Curie – Paris 6. We ran the abstraction layer on a laptop and, to evaluate the impact of mobility, we moved it from point A to point B at an average speed of  $\sim 6$  cm/s. Such a low speed has been intentionally chosen for the round-robin algorithm to run many times at each location. Fig. 5 shows the RSSI of two mini PCs (also running the abstraction layer) as measured from the laptop. Beacons are sent at 60 mW. The received signal strength of both mini PCs clearly increase when getting closer and decrease when moving away. Finally, Figs. 6, 7, and 8 show snapshots of the abstraction layer running in console mode on the laptop. These are just representations of the results of the curves at times  $t = \{65 \text{ s}, 87 \text{ s}, 155 \text{ s}\}$ .

## V. CONCLUSION

In this paper, we proposed an abstraction layer to help wireless protocol designers retrieving information from the neighborhood and the quality of the links. This tool applies an active measurement scheme that allows obtaining specific information that cannot be obtained with existing solutions, such as signal propagation information, estimation of packet error rate, and minimum transmit power. We showed through a

simple measurement study in a real environment that this tool allows collecting significant information on the neighborhood in a built-in fashion.

This abstraction layer stands as a building block to implement wireless multi-hop protocols, and is part as a prototyping environment that allows researchers to implement and evaluate simply and quickly their proposal in real conditions. We are currently developing such a prototyping system that will be made available to the community and used for evaluating new communication architectures such as the Radio Internet ([www.ist-wip.org](http://www.ist-wip.org)).

Future work also includes bringing modularity to the abstraction layer and to provide the designer with more cross-layer metrics and functionalities.

## REFERENCES

- [1] M. D. de Amorim, M. L. Sichiitiu, F. Benbadis, Y. Viniotis, and S. Fdida, "Dissecting the routing architecture of self-organizing networks," *IEEE Wireless Communications*, vol. 13, no. 6, p. 98.
- [2] I. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," *Computer Networks - Elsevier Science*, vol. 47, no. 4, p. 445.
- [3] M. E. M. Campista, I. M. Moraes, P. M. Esposito, A. A. Jr., D. O. Cunha, L. H. M. K. Costa, and O. C. M. B. Duarte, "The ad hoc return channel: a low-cost solution for brazilian interactive digital tv," *IEEE Communications Magazine*, vol. 45, pp. 136–143, Jan. 2007.
- [4] 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 *SenSys*, 2005, pp. 76–89.
- [5] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An operating system for wireless sensor networks," in *Ambient Intelligence*. Springer-Verlag, 2004.
- [6] 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 *USENIX Annual Technical Conference, General Track*, 2004, pp. 283–296.
- [7] Crossbow. Intel stargate. [Online]. Available: <http://www.xbow.com>
- [8] C. M. T. Calafate and P. Manzoni, "A multi-platform programming interface for protocol development." in *PDP*, Genova, Italy, Feb. 2003, pp. 243–249.
- [9] M. Youssef. MAPI: An API for wireless cards under linux. [Online]. Available: <http://www.cs.umd.edu/moustafa/>
- [10] Wireless Tools for Linux. [Online]. Available: [http://hpl.hp.com/personal/Jean\\_Tourrilhes/Linux/Tools.html](http://hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html)
- [11] H. Aiache, V. Conan, J. Leguay, and M. Levy., "Xian: Cross-layer interface for wireless ad hoc networks." in *Mediterranean Ad Hoc Networking Workshop*. Lipari, Italy, June 2006.
- [12] MadWiFi, Multiband Atheros Driver for WiFi. [Online]. Available: <http://madwifi.org/>
- [13] IPW2200, Intel PRO/Wireless 2200BG Driver for Linux. [Online]. Available: <http://ipw2200.sourceforge.net/>