

Neighborhood Monitoring and Link Assessment in Wireless Networks

Fehmi Ben Abdesslem, Luigi Iannone
Marcelo Dias de Amorim, Serge Fdida
LIP6/CNRS Laboratory
Université Pierre et Marie Curie – Paris VI
{fehmi,iannone,amorim,sf}@rp.lip6.fr

Ignacio Solis
Katia Obraczka
Computer Engineering Department
University of California, Santa Cruz
{isolis,katia}@cse.ucsc.edu

ABSTRACT

Despite the existence of an impressive number of wireless communication protocols, few have been tested under real conditions. This is mainly due to the difficulties found during the implementation phase. In this paper, we introduce *Warp*, a framework for fast prototyping of wireless network protocols. *Warp* provides a set of simple building blocks that implement common functionalities needed by a wide range of wireless protocols.

1. INTRODUCTION

Because of the countless challenges they pose, wireless networks have been the topic of extensive research. Nevertheless, most of the theoretical solutions proposed so far have seldom achieved the state of fully-functional implementation. The main reason for such a lack of practical realization is that implementation is a tough task. Although some efforts have already been done by the research community to bridge this gap, they are specific to wireless sensor networks [1, 2]. To our knowledge, nothing has been proposed to common networks such as IEEE 802.11.

In order to facilitate experimenting novel solutions, we propose *Warp*. *Warp* is a tool aiming at minimizing the gap between theoretical solutions and practical implementations in the context of wireless networks. The main goal of *Warp* is to help designers observing the behavior and performance of a wireless protocol *under real conditions* and with *little programming effort*. Prototypes implemented with *Warp* precede a final (optimum) implementation; the idea is to obtain an instantiation of the system that may not be optimized, but that is fully functional and complete.

The remainder of this paper is organized as follows. We introduce *Warp* in Section 2 and describe its two main components in details in Sections 2.1 and 2.2 before concluding the paper in Section 3.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Sigcomm'06, Sep. 11–15, 2006, Pisa, Italy.

Copyright 2006 ACM 1-59593-004-3/05/0005 ...\$5.00.

2. WARP OVERVIEW

Warp is composed of two main components: (i) *Warp library*, which provides an API of high-level primitives for sending and receiving data, as well as for retrieving information about the network; (ii) *Warp engine*, which relies on a probe-based mechanism to provide real-time status about the neighborhood.

The designer interacts only with the *Warp library*. The *Warp library*, in turn, communicates through the loopback interface with the *Warp engine*. This latter keeps an up-to-date list of the neighboring nodes and encapsulates/decapsulates the messages respectively sent/received by the library. Although we briefly present the *Warp library*, in this paper we focus more on the engine.

Warp is currently available on Linux, but we expect to port it to other operating systems. We deployed a small test-bed that includes mini-PCs and laptops running *Warp* on Red Hat 9 and Fedora Core 3 distributions, and using off-the-shelf heterogeneous 802.11 cards with power control features.

2.1 Warp Library

The *Warp library* provides a set of high level routing-oriented programming functions for different languages (implemented so far on C/C++ and Perl). The designer can use these functions without caring about sockets, communication setup, addresses, and other such details.

The library communicates with the *Warp engine* with simple request/reply mechanism. The *Warp library* primitives can be easily integrated into the prototype being developed. In practice, the *Warp library* is a simple couple of files to be included in the code of the communication algorithm being prototyped. For the sake of clarity, hereafter we only list the most important primitives:

- **Warp_Neighbors()**: Returns the list of neighbors in range, along with 2-hop neighbors and the statistics concerning the links' quality.
- **Warp_Send(Data, ID, Pwr)**: Sends a message containing *Data* to *ID* at the power level defined by *Pwr* (in mW).
- **Warp_Send_Broadcast(Data, Pwr)**: Sends a broadcast message containing *Data* at power level *Pwr*.
- **Warp_Receive()**: Checks if a neighbor message has been received by the daemon; if so, then return the message.

```

===== Neighbor List for node BOB =====
=====
2F6D Active 00:40:96:A9:2F:6D Beacon period : 10000 JOHN
Weakest beacon received by this neighbor : 1 mW
-----
1mW Active @9860 R O S 9 B 9 [0.015848932 nW (-78 dBm)] 4/5
12mW Active @9861 R O S 9 B 9 [0.079432823 nW (-71 dBm)] 4/5
21mW Active @9863 R O S 9 B 10 [0.199526231 nW (-67 dBm)] 5/5
29mW Active @9865 R O S 9 B 10 [0.501187234 nW (-63 dBm)] 5/5
60mW Active @9856 R O S 8 B 10 [1.995262315 nW (-57 dBm)] 5/5

2hop-Neighbors : ALICE (1 mW, -55 dBm) JACK (1 mW, -62 dBm)
=====

30A7 Active 00:40:96:A9:30:A7 Beacon period : 10000 ALICE
Weakest beacon received by this neighbor : 1 mW
-----
1mW Active @9857 R O S 9 B 9 [0.158489319 nW (-68 dBm)] 5/5
12mW Active @9859 R O S 9 B 10 [0.794328235 nW (-61 dBm)] 5/5
21mW Active @9860 R O S 9 B 10 [1.995262315 nW (-57 dBm)] 5/5
29mW Active @9862 R O S 9 B 10 [3.981071706 nW (-54 dBm)] 5/5
60mW Active @9863 R O S 9 B 10 [12.58925412 nW (-49 dBm)] 5/5

2hop-Neighbors : JACK (1 mW, -49 dBm) JOHN (1 mW, -56 dBm)
=====

```

Figure 1: Example of the information list provided by the Warp engine for node whose ID is “BOB.”

2.2 Warp Engine

The Warp engine is a daemon process. Its main goals are: (i) to respond to the requests of the Warp library with a regularly updated list of neighbors in range and some associated information about link quality; (ii) to encapsulate and de-encapsulate messages (respectively sent and received); (iii) to implement the entire neighborhood monitoring mechanism.

In the following, we detail how the Warp engine gathers information about the neighborhood and link quality.

2.2.1 Beacons sending mechanism

To build and to keep updated the list of neighbors, each node running Warp sends broadcast beacons periodically at different transmit powers in a round robin fashion. The values of the transmission power are customizable and depend on the characteristics of the wireless interface.

Upon the reception of a beacon (and ideally of a sequence of beacons), various statistics can be derived, eventually using the information included in the beacon packets. For instance, a node can deduce the minimum transmit power that the neighbor should use to send messages to it (the minimum **Transmit Power** among all the received beacons).

2.2.2 Feedbacks sending mechanism

A second type of packet is used by the Warp engine to retrieve statistical information about neighbors and links quality. These packets notify to each neighbor the transmit power of the weakest beacon received. The best received signal strength measured is also sent in these feedback packets to better characterize the link quality.

Although they are destined for a single neighbor, feedback packets are broadcasted and thus received by every neighbor. Broadcasting packets gives two-hop information about link quality and uses the radio medium exactly the same way a unicast transmission does. The only overhead incurred is due to the lower bit rate used by broadcast transmissions.

2.2.3 Information provided by the Warp engine

Information returned by the Warp engine (through the

`Warp_Neighbors()` primitive of the Warp library) includes for example the transmit power used for each beacon received, the received signal strength measured, or the rate of received beacons. Figure 1 shows a snapshot of the information provided by the Warp engine console running on the node named Bob. Basically, Bob has two active neighbors, John and Alice, and is able to receive all beacons sent by both. Nevertheless, in average, Alice’s beacons are better received than John’s ones. We can also see that Jack is a 2-hop neighbor, both by means of Alice and John. In the figure, Bob received 4 beacons out of the 5 beacons sent by John at 1mW, and the last one was received with -78dBm .

3. CONCLUSION AND ONGOING WORK

The aim of our research is to provide a new evaluation tool to designers by taking into account real world properties. Warp, our proposal, is a ready-to-use fast-prototyping environment. It works with off-the-shelf wireless cards and requires no specific end-system characteristics. Warp eases the development of prototypes, bridging the gap between theoretical solutions proposed for wireless communications and their practical implementation.

Besides the ongoing work to allow Warp running on different system (other than Linux), we are now studying and benchmarking the tradeoffs between the ease-of-use and the overhead incurred by Warp (mainly in terms of delay and throughput).

4. REFERENCES

- [1] 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.
- [2] 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.