A Real-World Framework to Evaluate Cross-Layer Protocols for Wireless Multihop Networks

Yves Igor Jerschow Björn Scheuermann Christian Lochert Martin Mauve

Institute of Computer Science
Heinrich Heine University Düsseldorf
Düsseldorf, Germany
yves.jerschow@uni-duesseldorf.de
{scheuermann, lochert, mauve}@cs.uni-duesseldorf.de

ABSTRACT

The MAC layer implementation of today's commodity 802.11 wireless network devices cannot easily be changed. But many cross-layer protocols for MANETs rely on a modified MAC layer. Therefore it is hard to test such protocols in real world environments. We propose to use a sensor network platform, the ESB sensor nodes, for this purpose. We present a software framework to make cross-layer protocol implementations and methodical experimentation feasible. The framework consists of software tools and modules for many frequently occurring tasks. It provides an extended link layer that increases the flexibility for protocol implementations on higher layers and it enables multihop communication on the ESB nodes by a network layer with static routing. An experimenter is supported by mechanisms to deploy routing tables, to gather network topology information and to obtain packet logs from all network nodes. We also give some experimentation results from an implementation of a cross-layer protocol using the framework.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Protocol verification; C.2.1 [**Network Architecture and Design**]: Wireless communication; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Experimentation, Measurements, Verification, Performance, Algorithms

Keywords

wireless networks, ad hoc networks, experimentation, real-world implementation, real-world protocol testing, cross-layer, framework

1. INTRODUCTION

Given the well known limitations of network simulation, real experiments should be a major building block of any MANET protocol evaluation. But the vast majority of the devices used for today's experimental MANET setups use IEEE 802.11 wireless LAN adapters. For those a large part of the MAC functionality is realized very close to the hardware, in the (proprietary) firmware. This makes modifications to the MAC layer for real-world experiments with cross-layer functionality nearly impossible, and this in turn prevents many protocol designs from being verified in a real setup.

When looking for a way to overcome these difficulties we came across the ESB sensor nodes (Figure 1). These relatively inexpensive devices were developed at the Freie Universität Berlin as part of the ScatterWeb project [**?**]. They are intended to serve as a testbed platform for wireless sensor networks. ESB nodes are battery-powered and equipped with a collection of sensors and a wireless interface. For the purposes of MANET protocol testing however, their main advantage is the open firmware, which allows modifications to every part of the software, down to the manipulation of each single bit transmitted on the wireless medium. Moreover, the modifications can all be made in a clearly structured code base. In short, the hardware platform and the tools provided allow the implementation of all kinds of changes to backoff mechanisms, cross-layer callbacks and other interaction mechanisms on the MAC layer and above. Of course, there are other sensor network platforms with similar features, and our approach could be transferred to those.

However, there is a problem with MANET protocol testing on sensor nodes: even in a small experimental setup with only a couple of nodes it is hard to gather the desired performance data, or even just to observe the protocol behavior. For example, no logging facility is available.

To overcome these problems and to enable the testing of cross-layer MANET protocols on the ESB sensor nodes, we have developed a MANET protocol testing framework for the ESB platform. It allows to set up experiments and to gather the result data. Thereby the ESB sensor network platform is transformed into a cross-layer testbed platform for MANET protocols.

Of course the non-802.11 compatible physical layer of the ESB nodes does not allow for a direct performance comparison to, e. g., 802.11-based networks. But its basic properties are similar to most wireless interfaces: a single-channel transceiver and a CSMA/CA MAC are used, carrier sensing is not possible while transmitting. As in many other sensor network platforms, an RFM TR1001 radio [**?**] is used that works at a fixed data rate of 19.2 KBit/s in the 868 MHz band. Thus, the ESB nodes have a lot in common with

Figure 1: An ESB sensor node.

other wireless devices. And although the absolute numbers are different, the most important step is to show that a protocol exhibits the desired general behavior in a wireless multihop network. The results can then guide further protocol refinements, or can motivate the much larger effort necessary to bring a certain modification closer to wireless interfaces with higher data rates.

As mentioned before, our work is based on the stock ESB firmware. We augmented it with a number of new features. For example, they comprise a network layer for multihop communication and some tools to support the experimenter. In addition to these completely new components, we have modified the link layer in order to increase its flexibility.

The remainder of this paper is structured as follows. In section 2 we will first review some related work. We will then describe our framework and the algorithms used to realize it in the resource efficient way required by the resource constrained hardware platform. How the ESB sensor nodes are programmed and controlled when experiments are conducted is described in section 3. Some modifications make the standard ESB link layer more flexible in order to support heterogeneous protocol demands. They are presented in section 4. Since for our own experiments we need only static routing, we have not implemented a fully fledged MANET routing protocol. Instead, our framework comprises a module for static routing that allows for user-defined routing tables. It is described in section 5. New routing tables can be deployed automatically in the network using a mechanism detailed in section 6. Also an automated discovery of the network topology is provided; this can support both routing table creation and verification of experimental setups. The corresponding mechanism is described in section 7. Finally, a last module of our framework, covered in section 8, provides a logging facility. The packets received, sent and overheard in all the nodes can be logged. This is necessary for a well-founded result analysis. In section 9 we show some experimental results obtained with our testbed and compare them to corresponding simulations; thereby we demonstrate the practical feasibility of our approach. We then conclude our paper in section 10.

2. RELATED WORK

In [**?**] Bernasconi et al. describe a hardware architecture they have developed in order to be able to perform real world experiments with their modified MAC backoff scheme. Such a tailored hardware design obviously provides a high degree of flexibility. However, their tailored hardware is not widely available, and can thus not easily be used by other researchers who intend to perform real world experiments with their protocols. Since the hardware platform is not standardized it is hard to reproduce experimental results or to make direct comparisons to possible implementations of other approaches. The ESB sensor nodes in contrast are commercially available and can be ordered at a reasonable price in larger quantities. Additionally, the nodes are small, autonomous, and battery-powered. Hence the deployment of the nodes for an experiment is quite simple because neither a power supply nor a connection to a PC is required.

In [**?**] Heissenbüttel et al. describe a real world implementation of a geographic routing protocol. They also encountered the difficulty that the MAC protocol cannot be changed to meet the needs of the proposed MAC layer. This would have been possible on a sensor-network based testbed.

There is some literature describing experiments with alternative MAC protocols for sensor networks, e. g. in [**?**, **?**]. We, in contrast, look primarily at protocols for MANETs or Wireless Mesh networks. In those networks, the applications and their requirements are very different. We just use the sensor nodes as a testbed platform.

The ESB sensor nodes are used in various sensor network testbeds with applications that are specific to sensor networks. Examples of this work are [**?**, **?**, **?**].

In [**?**] Dunkels et al. present a prototype sensor networking platform using the ESB nodes with the Contiki OS [**?**]. The key features of their platform are an energy aware MAC layer and a TCP/IP protocol stack with sensor network specific extensions. It would have been an option to base our framework on Contiki instead of the standard ESB firmware. However, their approach provides a lot of sensor network specific functionality and is thus relatively complex. We therefore decided not to use Contiki as a basis, in order to make protocol implementations as straightforward as possible and to limit the amount of code to a minimum.

One of the central mechanisms in our framework is a topology exploration algorithm, which we use for the preparation of experiments, the verification of experimental setups and the automated generation of routing tables. The discovery of the current network topology is also a central element in other areas, especially in proactive wireless routing protocols like TBRPF [**?**] or OLSR [**?**]. However, the focus is different there since in the case of routing the topology information is disseminated in the background, trading off up-to-dateness against network load. The intention is to achieve a fast dissemination of topology information in the whole network with minimum effort. In contrast, for our purposes a fast and reliable collection of the complete topology information at *one central point* is necessary.

In [**?**] Deb et al. describe a topology discovery algorithm for sensor networks. Their algorithm is meant to be used for network management purposes like network state retrieval and as a basis for the setup of efficient data dissemination and aggregation paths. In their approach, only an approximation of the network topology is collected. However, our topology discovery algorithm is in some aspects similar to their "Aggregated Response" algorithm used for comparison purposes in the same paper. "Aggregated Response" explores the topology in parallel, using a flooding strategy. This can cause unintended interference effects like packet collisions due to the shared medium and thus lead to missed links if discovery packets collide. Our topology exploration algorithm serializes the topology discovery and thus eliminates interference effects within the network.

3. PROGRAMMING AND CONTROLLING THE ESB NODES

The base firmware and our protocol modules are written in C. The respective cross-compiler and debugging tools are freely available for several platforms [**?**]. Thus, no deeper assembler programming skills are required, and various protocols can be implemented really straightforward. The compiled binary module can be transferred to the nodes using a JTAG interface, which is connected to a parallel or USB port. This interface is also used for online debugging.

The nodes are operated via a terminal program. At least one node must hence stay attached to the serial port of a desktop computer. The others can be then controlled remotely. The user queries the nodes' state or configures various parameters by typing terminal commands. There are some predefined commands built into the standard ESB firmware, many more were added by us. For example, it is easily possible to display the routing table, to send a packet to a remote node or to set some timeout parameter in all nodes simultaneously by broadcasting the respective command. Since the nodes can be switched into a verbose mode, all the network activities can be monitored by following the debug messages displayed on the terminal. This makes debugging a lot easier.

The helper applications for the routing table transfer, logging and topology services also exchange data with the nodes over the serial interface.

4. LINK LAYER EXTENSIONS

The standard link layer in the ESB firmware allows the reliable single-hop transmission of data packets. Link layer acknowledgments and Automatic Repeat Requests (ARQ) are used for this purpose. This turned out to be a good foundation for our work. But since many protocols require more or different functionality, we extended the basic link layer functionality.

The ESB nodes allow for an adjustment of the transmission signal strength in 100 steps. In our framework, the signal strength can—upon demand—be individually adjusted for every single packet. The information on the signal strength to be used is stored with the packet in the queue. Adjusting the signal strength is essential, e. g., for topology control mechanisms. In addition to that, the modification of the signal strength has also been proposed in other contexts, for example to salvage packets if a link layer connection has just broken with normal signal strength [**?**].

Some protocols do not require link layer acknowledgments. Examples are passive acknowledgment approaches, as proposed, e. g., for the DSR routing protocol [**?**]. The number of retransmissions might also differ from packet to packet. To handle this with maximum flexibility, a way is provided to determine the use of link layer reliability mechanisms on a per-packet basis as well as a way to work with an individual number of ARQ packet retransmission attempts.

The link layer frame format used by our framework is shown in Figure 2. From the viewpoint of a cross-layer protocol developer this format is not to be understood as fixed, but rather as the basis for further, protocol-specific adaptions.

The format does not contain a field for the specification of the transmission power or the ARQ retry count to be used. Instead, this information is stored in the network layer packet header. This is because these values can optionally be predefined for the whole route by the original sender of a multihop transmission. If they were stored in the link layer header they would be lost after the first hop. It is of course also possible to use cross-layer protocol feedback to set or alter these values independently at each hop.

Figure 2: Link layer packet.

Figure 3: Network layer packet.

5. STATIC ROUTING

The main focus of our current work is on MAC/transport layer interactions. Therefore we were looking for a way to eliminate routing protocol effects from our experiments. The intention is to be able to distinguish routing protocol influences from inherent MAC and transport layer issues. Our solution to this challenge is to use a static topology with static routing. To support this on the ESB platform, a static multihop routing module—including an appropriate handling of routing tables—is required. This is a valuable testbed function for anyone seeking to distinguish routing effects from other, e. g., shared medium related, effects.

The static routing tables for a given experimental scenario can be easily crafted by writing a routing configuration file for the network in a simple, text-based syntax. For each node and destination address, the corresponding next hop is defined. Destination addresses with the same next hop may be aggregated to keep the configuration file concise. A default gateway can be specified for each node. A helper application parses the routing configuration file and transfers the routing tables to the nodes.

Given some network topology—either manually defined or automatically discovered—the optimal routing tables can also be computed automatically by another helper application, using the Dijkstra algorithm.

To economize on the very limited main memory the static routing table is stored in the flash memory of the node besides the firmware code. Unlike the also available EEPROM chip, the flash memory allows fast and direct access by means of a pointer.

Figure 3 shows the network layer packet format. Again, this is only to be understood as a basis for further, protocol-specific adaptions. The 16 Bit ID allows to identify each datagram when evaluating log files. The service flags byte carries the requests for special datagram handling that have been mentioned before, like modifications of the used signal strength, the number of link layer retransmissions or prioritized packet handling.

6. DISTRIBUTION OF ROUTING TABLES

In order for the static routing module to work, each node needs to know its routing table. Since these tables are likely to change often from experiment to experiment—depending on the network topologies used—deploying a new set of routing tables to the nodes is a common task. To ease this task, a mechanism has been implemented that uses the routes given in the set of routing tables themselves to supply each node with its routing table in an efficient way.

With this mechanism we intend to provide a means to distribute routing tables while leaving the nodes in-place and without visiting each node individually. The only prerequisites that need to be fulfilled to make this work are that the network is connected, and that the routing tables that are to be deployed are valid, i. e., that

they contain working routes. These routes are used to distribute the routing table information in the network.

In our framework, one node—the so-called *master node*—is connected to a desktop computer and serves as the source for the routing table distribution. The routing tables are read from a file on the desktop computer, where they are stored in an easy-to-read, text-based format. A special tool reads this file and transfers the routing tables in a compressed format to the master node.

The routing paths from this master node to all other nodes form a tree. This routing tree is used to distribute the routing tables. Along each edge of the tree the routing tables for all nodes in the specific subtree are transmitted via reliable unicast, packed together in one *routing table package*. Each node in the tree that receives such a package extracts its own table. The information contained therein can then be used to pass the other tables further on towards the tree's leaves. Therefore all routing tables finally reach their respective nodes. This is schematically depicted in Figure 4.

One important benefit of this scheme is that it is not necessary to have any routing information available in the nodes prior to the distribution of the tables. It is also not necessary to use any routing protocol in order to achieve the intended distribution—to have a set of valid routing tables available at the master node is sufficient, because it can be utilized for its own distribution.

If optimal routes are used for the routing in the experiment, the distribution paths will also be optimal. Thus the distribution is achieved with minimal effort. Assume that the routing tables define hop-count minimal routes. Then the tree used for distribution is the minimum hop-count routing tree with the master node as a source. Thus all the routing tables will be distributed along the minimum hop-count path to their respective destination node. Note that this property does not depend on this particular notion of optimality; it would also hold for any other (sensible) definition of "optimal".

However, it is possible that experiments will be conducted where intentionally bad or non-existing links are used, or where the network is intentionally partitioned by leaving out working links. For these special cases it is also possible to define specific *deployment paths* in the routing tables. This additional information is not used for the routing—it is not even stored in the nodes—but for the deployment of the routing tables. Therefore links that are intentionally left unused in the experiment can nevertheless be used for routing table deployment.

Figure 4: Routing table distribution in an example network topology.

7. TOPOLOGY EXPLORATION

When an experiment with wireless communication devices is set up it is usually hard, but necessary to verify that the actual topology matches the intended one. The topology exploration module of our framework makes this verification easier. It is able to gather topology information from the network and deliver it to the user. The topology information can also be used to generate routing tables in the format required for the routing table distribution tool from section 6, either to be used directly or as a basis for a user-defined routing.

The topology exploration algorithm takes care to report the topology of the network as accurately as possible. In particular, the exploration of the topology is serialized. This has the advantage that the probability of lost packets due to collisions and thus of links remaining undetected is minimized. In the proposed algorithm there is always exactly one node performing a neighbor search. Flooding the network, where many collisions can occur and thus many packets are lost, is explicitly avoided.

A token-based depth-first search strategy is employed by the topology exploration algorithm. A *master node* is connected to the user's desktop PC and starts the topology exploration process. This node first uses a repeated broadcast ping request to find its neighbors. All nodes that hear such a broadcast reply with a pong packet after a random backoff period, to minimize packet collisions. The searching node keeps track of the answering neighbors and builds its neighbor table. The number of ping replies successfully received from each neighbor is also stored and can be used as a metric for the link quality.

The master node then generates a token. After the generation of the neighbor table the node passes this token on to one of its neighbors. Then it waits until it returns from this neighbor, and passes it on to the next one, and so on. A node receiving the token for the first time performs a neighbor search as described above and then passes the token on to *its* neighbors in turn, one after the other. The token passing is implemented reliably using ARQ.

When the token returns from the last neighbor on the list, it is handed back to the node it originally came from. In this case, the neighbor table of the current node plus all the neighbor tables that have potentially been received by this node from its neighbors are attached. If a node that has already performed a neighbor search receives the token again, it is immediately sent back, without any neighbor tables being attached. Likewise, if the table of a neighbor node is already known, the token does not need to be passed to this neighbor.

The algorithm ensures that every node performs a neighbor discovery exactly once, and that all the neighbor tables eventually arrive at the master node. From there they are transferred to the desktop computer and can be processed further, for example to draw a topology graph of the network or for the generation of routing tables. Algorithm 1 is a pseudocode version of the algorithm.

The described version of the algorithm supports only bidirectional links. It can easily be extended to detect also unidirectional links. Every node can build a list of nodes from which ping requests have been received. After the token has returned to the master node, these lists are collected. This could, for example, be accomplished by letting the token traverse the network a second time, without a new neighbor search, but just collecting the lists of received ping requests. If a node A has received ping requests from another node B, but does not appear in B's neighbor list, then there exists a unidirectional link from B to A. We have not yet implemented this extension, because we use only bidirectional links for the routing and therefore do not necessarily need data on unidirectional links.

```
 1: t1 ← receive token
 2: N ← neighbor search
 3: L ← {N}
 4: for all n ∈ N \ {t1.received_from} do
 5:    if no table of n in L then
 6:       pass token to n
 7:       repeat
 8:          t2 ← receive token
 9:          if t2.received_from ≠ n then
10:             return token to t2.received_from
11:          end if
12:       until t2.received_from = n
13:       L ← L ∪ t2.attached_tables
14:    end if
15: end for
16: return token plus L to t1.received_from
17: loop
18:    t ← receive token
19:    return token to t.received_from
20: end loop
```

Algorithm 1: Topology exploration.

8. THE LOGGING SERVICE

When experiments with mobile ad-hoc network protocols are conducted, it is not only necessary to enable an easy implementation of the protocols that are to be tested, and it is also not sufficient to support the experiment itself. In addition it is important to enable the collection of the result data in a form that can be understood easily, processed automatically, and stored permanently for future reference. While the logging of network events is quite straightforward on more powerful hardware, there is no default logging service on the ESB sensor nodes. Logging for the ESB nodes is a challenging task especially because of the limited resources: there is only very limited storage space available for log entries.

We have developed a full-featured logging mechanism that allows a cross-layer recording of packet traces. Immediately before a packet reception and transmission, as close as possible to the physical layer, the time stamp of the event is recorded. This ensures maximum time stamp accuracy. Furthermore, selected header fields from the different protocol layers are contained in the log entries, e. g., the sender and receiver on the link layer or the source, destination, hop counter, and the datagram ID on the network layer. The logging service can be configured to record incoming, outgoing and even overheard foreign packets *(promiscuous mode)*. When monitoring outgoing packets with the reliable data transfer at the link layer enabled, the log record is extended by two entries which are not part of any header: the number of retransmissions and a boolean value indicating whether the packet was finally acknowledged by the receiver or not. This log provides plenty of information for protocol analysis, debugging and performance evaluation.

The log records are written to the 64 KB EEPROM chip. It is the largest storage device available and is not occupied by the system during normal operation. But of course, 64 KB of traces per node is still very limited. To account for this, the storage space for each log entry is calculated bitwise, considering the dimension of the sensor testbed and the time gap during which the network traffic shall be logged. The utilized nodes are enumerated continuously from 1 to n and these numbers are used at the link and network layer for addressing. In a testbed consisting of n nodes only $\lceil \log(n+1) \rceil$ bits are needed to represent an address instead of the 16 bits used in the protocol headers.

The nodes' clock offers a resolution of 1 ms and represents the time and date in a 32 bit integer. However, instead of wasting valuable storage space, it is sufficient to count the number of milliseconds passed since the activation of the logging service. A logging time span (i. e., an experiment duration) of one hour requires to reserve only 12 bits for a time stamp. Considering the measures to economize on space, the EEPROM can hold several thousand log records enabling a detailed analysis of complex scenarios.

To easily compare the different log records monitored by the nodes participating in the experiment setup synchronous time stamps are extremely helpful. Therefore, the nodes' clocks are initially synchronized: the master node is defined to be the reference time source. It synchronizes the other nodes to its clock by broadcasting its current time. The experimentally measured time gap for sending and processing the synchronization packet is eliminated.

Writing to the EEPROM is a blocking operation. If it is used inconsiderately, it might bar the node from sending or receiving packets for up to a couple of milliseconds. It is important to take care that the flushing of the trace data does not change the behavior of the node too much, because this could tamper the results of the experiment. In order to minimize the delays incurred by the logging service, we utilize an appropriately sized logging buffer in main memory. When this buffer is full, it is flushed to the EEPROM in a manner that allows the interrupt-driven physical layer to continue receiving or transmitting data.

After the experiment, a helper application transfers the log data to a desktop computer and creates a logfile. The format of this file is text-based, and it is thus much easier both to read and to parse (e. g., for performance analysis) than the compressed bit-field format used in the EEPROM of the ESB nodes.

9. EXPERIMENTS

We have done an implementation of a MANET cross-layer congestion control protocol currently under development, using the framework described in this paper. Here we present some of the obtained results. We do this to show the practical feasibility of our approach. Thus the details of the protocol itself are of subordinate importance, and we will not cover them in depth here.

We used a bidirectional chain topology for the experiments. It exists one sender at each end of the chain transmitting data to the respective other end of the chain. The chain has been set up by configuring the (static) routing tables appropriately. Figure 5 compares the end-to-end throughput obtained using a 802.11-like packet forwarding mechanism and our own in-network congestion control protocol (named CXCC). The left chart shows ns-2 simulation results from a 5-hop chain, the right chart testbed results from an equivalent setup. Both charts show how the throughput develops with increasing network load. In both cases the throughput with the 802.11-like packet forwarding drops immediately after a certain load threshold is exceeded, while CXCC's throughput remains stable.

The implementation of the CXCC protocol—requiring significant modifications to the common layering and in particular MAC and transport layer interaction—was possible with moderate effort. On a quantitative level the results of the ns-2 simulations and the real world experiments were dramatically different. This was hardly surprising given the vastly different radio layers. However, much more importantly, on a qualitative level both simulation and real-world experiments displayed a similar overall behavior. We view this as a confirmation that our approach of using the ESB sensor nodes as a basis for real-world cross-layer experiments is both valid and practicable.

(a) Ns-2 simulation results.

(b) ESB testbed results.

Figure 5: Comparison of simulation results and ESB testbed results of a cross-layer protocol.

10. CONCLUSION

In this paper we have presented a software framework on the basis of the ESB sensor network platform. This framework allows for the easy implementation and evaluation of cross-layer MANET protocols, including those with a modified MAC layer. In general, it is not possible to implement all the changes required by these protocols on a testbed based on IEEE 802.11 hardware. In contrast on our ESB-based testbed platform it is possible to realize nearly all kinds of modifications, and to evaluate protocol designs in small to medium-sized setups with reasonable effort.

We have described the functionality of our framework, extending the basic ESB platform. Our enhanced link layer allows for a reliable and unreliable single-hop packet transmission. A resource efficient logging facility makes debugging easier and is indispensable for the collection, documentation and evaluation of experimental results.

In addition to these low-layer extensions, a static routing agent provides multihop connectivity and therefore a means to test transport layer protocols in static networks, eliminating routing protocol influences. Convenient tools allow for an easy creation and deployment of routing tables, including an automatic exploration of a set-up topology.

Our framework has successfully been used to test a cross-layer transport protocol. In these experiments the proposed solution has proven its practical feasibility as well as the intended flexibility—the discussed cross-layer protocol requires major modifications to the way MAC protocols commonly interact with the higher layers, and an implementation on 802.11-based hardware would not have been possible. Based on this experience we believe that the discussed testing approach is of interest for many researchers, and can often serve as a basis for experimental verification of simulative results, where otherwise it is hardly possible to realize an implementation on today's wireless devices.

11. REFERENCES

[1] R. Bernasconi, R. Bruno, I. Defilippis, S. Giordano, and A. Puiatti. Experiments with an enhanced MAC architecture for multi-hop wireless networks. In *Proceedings of REALMAN 2005*, July 2005.

[2] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, 2003.

[3] The Contiki Operating System. http://www.sics.se/ adam/contiki.

[4] B. Deb, S. Bhatnagar, and B. Nath. A Topology Discovery Algorithm for Sensor Networks with Applications to Network Management. In *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking*, September 2002.

[5] A. Dunkels, L. M. Feeney, B. Grönvall, and T. Voigt. An integrated approach to developing sensor network solutions. In *Proceedings of the Second International Workshop on Sensor and Actor Network Protocols and Applications*, August 2004.

[6] Freie Universität Berlin, Computer Systems Telematics. ScatterWeb Project. http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net.

[7] M. Heissenbüttel, T. Braun, T. Roth, and T. Bernoulli. GNU/Linux Implementation of a Position-based Routing Protocol. In *Proceedings of REALMAN 2005*, July 2005.

[8] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*, volume 353, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[9] F. Klemm, Z. Ye, S. V. Krishnamurthy, and S. K. Tripathi. Improving TCP Performance in Ad Hoc Networks using Signal Strength based Link Management. *Ad Hoc Networks*, 3(2):175–191, 2005.

[10] mspgcc – GCC toolchain for MSP430. http://mspgcc.sourceforge.net.

[11] R. Ogier, F. Templin, and M. Lewis. Topology Dissemination Based on Reverse-Path Forwarding (TBRPF). RFC 3684, 2004.

[12] RFM TR1001 868.35 MHz Hybrid Transceiver Data Sheet. http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net/datasheets/TR1001.pdf.

[13] J. Schiller, A. Liers, H. Ritter, R. Winter, and T. Voigt. ScatterWeb – Low Power Sensor Nodes and Energy Aware Routing. In *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS 2005)*, January 2005.

[14] S. Schmidt, H. Krahn, S. Fischer, and D. Wätjen. A Security Architecture for Mobile Wireless Sensor Networks. In *First European Workshop on Security in Ad-Hoc and Sensor Networks (ESAS 2004)*, August 2004.

[15] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180, New York, NY, USA, 2003. ACM Press.

[16] T. Voigt, H. Ritter, and J. Schiller. Solar-aware Routing in Wireless Sensor Networks. In *9th International IEEE Symposium on Computers and Communications (ISCC)*, July 2004.

[17] W. Ye, J. S. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the IEEE INFOCOM*, 2002.