Simulation of Unidirectional Links in Wireless Sensor Networks

Reinhardt Karnapke
Distributed Systems/
Operating Systems group
BTU Cottbus - Senftenberg
Cottbus, Germany
karnapke@informatik.tu-cottbus.de

Stefan Lohs
Distributed Systems/
Operating Systems group
BTU Cottbus - Senftenberg
Cottbus, Germany
slohs@informatik.tu-cottbus.de

Jörg Nolte
Distributed Systems/
Operating Systems group
BTU Cottbus - Senftenberg
Cottbus, Germany
jon@informatik.tu-cottbus.de

Andreas Lagemann
Nanotron Technologies GmbH
Berlin, Germany
a.lagemann@nanotron.com

ABSTRACT

Experiments with wireless sensor networks have shown that asymmetric and unidirectional links are common. For this reason, we developed different routing protocols that can use unidirectional links, either implicitly or explicitly.

However, developing protocols that use unidirectional links is difficult, not least because of inadequate simulation support.

In this paper we present a simulation model based on OMNeT++ and MiXiM that we developed to simulate networks with unidirectional links and frequent link changes. We evaluate the developed simulation approach by comparison of results achieved by different routing protocols in simulation and in experiments with real sensor network hardware.

Categories and Subject Descriptors

I.6.3 [**Computing Methodologies**]: Simulation and Modeling

General Terms

Experimentation

Keywords

Simulation, Wireless Sensor Networks, Unidirectional Links, Routing

1. INTRODUCTION

Asymmetric and unidirectional links are common in wireless sensor networks. This has been proven in experiments with different hardware (e.g. Scatterweb ESB [18], XSM Motes [15, 16], Mica Motes [21, 22], Mica 2 Motes [6], Ez430-Chronos [10]).

The length of asymmetric and especially unidirectional links exceeds that of bidirectional links by far. This greater reach reduces the number of hops needed for multihop routing and, consequently, the number of communicating nodes as well as the overall energy consumption. Therefore, protocol designers should consider designing protocols that use unidirectional and asymmetric links [22].

Designing protocols for asymmetric and unidirectional links is not easy, though. Detecting them on the link layer and making them usable on the routing layer often induces a high overhead [11]. When protocols have been designed to detect and/or use asymmetric and unidirectional links (e.g. DEAL [2], ETF [15]), the protocol developers face the next challenge: Evaluation of their protocols.

The evaluation of protocols for sensor networks is usually twofold: simulations and experiments. Proving that the link detection scheme works in a real network is hard as it is impossible to prove that all existing links were detected. Unidirectional links can be artificially inserted by setting nodes to different transmission strengths. Then, it is possible to check if these links were discovered. The same method, setting different transmissions strengths to induce unidirectional links, is also used in some simulations.

Another property of wireless sensor networks is the instability of radio connections. Depending on a number of parameters (e.g. residual energy, antenna, deployment height, weather), links change more or less often, sometimes within minutes [10]. When these frequent link changes are taken into account, the evaluation of a link discovery protocol in real deployments gets even harder. Lots of parameters need to be considered, and it is never possible to be sure if all existing links have been measured. Therefore, simulations become more important.

In simulations, it is possible to evaluate protocols under controlled, repeatable circumstances. This enables a comparative evaluation of protocols, i.e., which protocol discovered more of the existing links.

Simutools 2014, March 17-19, Lisbon, Portugal

DOI 10.4108/icst.simutools.2014.254791

Unidirectional links are often realized using different transmission strengths for nodes in these simulations, resulting in stable unidirectional links. In reality, however, links are far from stable. When protocols that use asymmetric and unidirectional links are simulated, frequent link changes should also be taken into account.

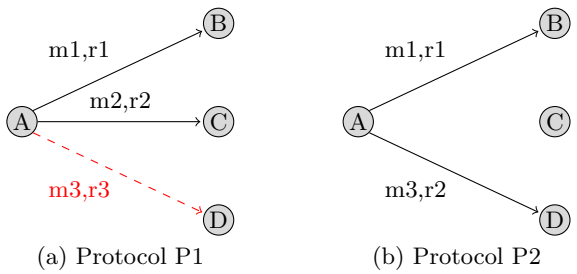
In previous work, we developed *Buckshot Routing* [14], a routing protocol for wireless networks with unidirectional links. During the development of Buckshot Routing (and a number of protocols developed thereafter) it became clear, that the existing simulation approaches had their limitations when unidirectional links were concerned. After we measured the link stability - or rather, the absence thereof - in a sensor network that we were going to use as testbed [10], we realized that not only the unidirectional links but also the instability of links had to be taken into account. Introducing stable unidirectional links by setting nodes to different transmission strength was not realistic enough. Therefore, we developed a new, matrix based approach.

In this paper we present this matrix based simulation approach and its implementation for the OMNeT++ discrete event simulator framework based on MiXiM, a model framework targeting wireless network simulations. It enables developers to evaluate, e.g., routing protocols in networks with unidirectional links and frequent link changes.

This paper is structured as follows: Our simulation framework is described in section 2, followed by a comparison between delivery ratios of routing protocols achieved within our simulations and those achieved in real world experiments using 36 sensor nodes. Related work is given in section 4. We conclude the paper in section 5.

2. SIMULATION FRAMEWORK

In many simulations, unidirectional links are realized by initializing nodes with different transmission strengths, e.g., one half of the nodes with transmission range X, the other half with 2X. This approach has the severe drawback that unidirectional links are static throughout the simulation, whereas they can become bidirectional or change direction in real sensor networks. Other approaches use a random number generator to decide if a connection exists only when a node tries to transmit a message. Even though this approach can be used to simulate a single protocol, drawing conclusions is hard as the logical (radio) topology is not always known. Moreover, comparative studies, which compare the results of two or more different protocols suffer a new problem: Different protocols are sometimes presented with different topologies, depending on the network load (Figure 1):



(a) Protocol P1 (b) Protocol P2

Figure 1: Influence of Random Numbers

Protocol P1 transmits three messages m1, m2 and m3. The links over which these messages are transmitted are evaluated based on random numbers r1, r2 and r3. Assuming r1 and r2 represent a connection and r3 represents a broken link, messages m1 and m2 reach the next node but message m3 is lost (Figure 1(a)). Due to protocol differences, protocol P2 does not transmit message m2, it only transmits messages m1 and m3. As the sequence of random numbers delivered by the random number generator is always the same, random number r2 is now used for the link over which message m3 should be transmitted, resulting in a connection between nodes A and D for protocol P2, whereas there was no connection for protocol P1 (Figure 1(b)).

To circumvent this problem, we use connectivity matrices which can be generated or measured in a real sensor network and fed into the simulation. To take the frequent link changes into account, we introduce time steps. When the simulation starts, an initial connectivity matrix represents the connections between nodes. This matrix is replaced by a new one with every time step. By using predefined matrices, a comparison of protocols with the same logical (radio) topology is possible.

2.1 Network Communication in OMNeT++ with MiXiM

We integrated our simulation model into the discrete event simulator OMNeT++ [19] (version 4.1) with the MiXiM [8] extension. OMNeT provides a simulation framework for generic networks. The MiXiM extension enables simulations of MANET or wireless sensor network (WSN) communication.

In MiXiM, nodes are connected using so called *network interface cards* (NICs). A NIC consists of MAC- and physical layer. Only if NICs are connected, communication between nodes is possible at all. Additionally, an AnalogueModel and a Decider are included. The AnalogueModel is used to attenuate the transmitted signals. The Decider decides whether a message has been correctly received at the destination. AnalogueModel and Decider are connected through, and initialized by, the physical layer.

When nodes want to transmit messages, they are handed to the chosen MAC layer which may delay them (e.g. TDMA) or evaluate the state of the physical layer first (e.g. CSMA). Once the MAC layer hands the message to the physical layer, the duration of message transmission is calculated and the message is appended in the receive queue of all nodes whose NICs are connected.

Due to the fact that collisions occur at the receiver, the messages in their receive queues are evaluated when simulation time has progressed to their reception time. Then, the Decider checks if multiple messages have been received at the same time (i.e., a collision occurred) or the channel was free. In the first case, the message is deleted. In the second case, if the signal to noise ratio was good, the message has successfully been received and is handed to the upper layers.

2.2 Integration into MiXiM

To integrate our model into MiXiM, a number of modules needed to be modified:

- MAC layer
- Physical Layer
- AnalogueModel

- Decider
- ConnectionManager

We use an empty MAC implementation, because we want to evaluate the influence of unidirectional links on routing protocols, not their interaction with the MAC layer. Therefore, the *NullMAC* always hands messages to the physical layer directly. If, however, the influence of different MAC layers was to be evaluated, it would be easy to replace the *NullMAC* with different MAC protocols.

The *NullPhy* represents the connection between *NullAnalogueModel* and *NullDecider* and is used to initialize both.

The *NullDecider* ignores the signal to noise ratio and always hands the message to the upper layer. Please note that this is possible because the decision whether a message is received or not is made based on the matrices (see below).

While the connections between NICs are static in most simulations, we use an extended ConnectionManager that enables changing links at runtime. The class *DynamicConnectionManager* is an extension of the UnitDisk ConnectionManager supplied by MiXiM. The range used for the unit disk graph is set to engulf the whole area simulated, theoretically enabling transmission between all nodes. The *DynamicConnectionManager* offers four additional methods. A link between two nodes can be added or removed, in one direction or in both, during the simulation (Listing 1).

```
class DynamicConnectionManager
    : public UnitDisk
{
 protected:
 virtual void initialize(int stage);

public:
 /**
 * Remove all connections between NICs
 * with NIC ID from and to.
 * @param from One side of the link(s)
 * @param to The other side of the link(s)
 * @return 0 if no links existed
 * 1 if only a unidirectional link existed
 * 2 if a bidirectional link existed
 */
 char disconnectBi(cModule* from,cModule* to);

 /**
 * Deletes a unidirectional link.
 * @param from
 * Start of the link that will be deleted
 * @param to
 * End of the link that will be deleted
 * @return
  *True if the link was deleted,
 * false if the link did not exist
 */
 bool disconnect(cModule* from,cModule* to);

 /**
 * Connects two NICs with a bidirectional link
 * @param from One side of the link(s)
 * @param to The other side of the link(s)
 * @return
 * 0 if a Bidirectional link already existed
 * 1 if only a unidirectional link existed
 * 2 no link existed
 **/
 char reconnectBi(cModule* from,cModule* to);

 /**
 * Establishes a connection from "from"
 * to "to" but not the other way around.
 * @param from the originator of the link
 * @param to the destination of the link
 * @return True if the link was created
 * false if the link has already existed
 */
 bool reconnect(cModule* from,cModule* to);

};
```

Listing 1: DynamicConnectionManager: additional Methods

With the *DynamicConnectionManager*, all tools that are needed to implement the matrix based connectivity model are available. The module *MatrixSwitchModule* (Listing 2) periodically changes links. The connectivity matrices are gathered or generated before the simulations start and stored in files. To reduce the size of the files, only link changes are stored. A link change is noted as a tuple (from, to, type) where *to* and *from* are the IDs of the nodes which are affected by the change and *type* specifies whether the link appears or disappears. Please note that a bidirectional link that disappears requires two entries in this notation. As the number of link changes may vary a lot between subsequent matrices, the *MatrixSwitchModule* first reads the number of changes from the file before the changes themselves are read and enacted.

```
class MatrixSwitchModule:public BaseModule{

public:
 virtual ~MatrixSwitchModule();
 virtual void initialize(int stage);
 virtual void handleMessage(cMessage* msg);
 virtual void finish();
 void switchLinks();

 int numInitStages() const {
  return 3;
 }

protected:
 // timer
 cMessage* rescheduleTimer;

 DynamicConnectionManager* conMan;

 // state variables
 int currentMatrix;
 int lastMatrix;
 int numberOfNodes;
 int numNodesRead;
 int numConnections;

 int originator;
 int destination;

// 1 = reconnect, 0 = disconnect
 bool typeOfChange;

 std::ifstream* file;
};
```

Listing 2: The MatrixSwitchModule

All link changes happen in the method *switchLinks()*. It is called for the first time by the *initialize(int stage)*-method when the stage has reached the value two. Waiting for two stages is necessary to ensure that the NICs have already been created. Then, a self message is created which schedules the *MatrixSwitchModule* again after a certain time. The message is received by the *handleMessage(cMessage* msg)*-method, which calls *switchLinks()*, increases the count for already processed matrices, and checks if there are more matrix changes planned. If there are further changes, it creates another self message. Otherwise, the last switch of matrices occurred and the module becomes inactive.

3. EVALUATION

To evaluate our approach, we measured the delivery ratio of routing protocols in simulation and on real hardware and compared the results.

We generated connectivity matrices which were meant to resemble the conditions described in [10]. In each of these matrices, a (directed) link from node A to node B exists with a probability of α/d^6 where d is the distance between node A and node B. The inverse link, from node B to node A exists with the same probability. Therefore, the link is bidirectional with a probability of $(\alpha/d^6) \times (\alpha/d^6)$, unidirectional (in any one direction) with $\alpha/d^6 \times (1 - (\alpha/d^6))$ and non existing with $(1 - (\alpha/d^6))^2$. The quotient (d^6) reflects the attenuation induced by the distance between nodes while α represents the probability that a link between geographically adjacent nodes exists.

Please note that the attenuation effect seems to be stronger than what is used in literature (d^6 vs. d^4), but the resulting connectivity graphs resemble those measured in [10] more closely. This stronger effect is possibly caused by the usage of low quality antennae.

Nodes were arranged on a regular grid to reflect application scenarios which need area coverage, e.g., vehicle tracking. As all nodes were arranged on a grid, nodes that are directly above, below, right, or left of a node are called direct neighbors and their distance was defined as 1. α was varied between 0.9, 0.95 and 1, and for each value of α ten sets of matrices with different seeds for the random number generator were generated.

Please note that due to the fact that the matrices were generated randomly, there is no guarantee that there always was a path from sender to destination. Therefore, no upper limit can be calculated, but *Flooding* is used as reference protocol: The number of application messages delivered by*Flooding* is taken as 100% and the delivery ratio of all other protocols is calculated accordingly.

The implemented application represents a sense-and-send behavior that is often found in sensor networks: All nodes within the network want to transmit all their messages to the same destination.

In each simulation, each node wants to transmit a total of 110 messages to the sink. After the initialization phase of the network, one message is transmitted every 100 milliseconds. To ensure that route discovery is finished, the logging remains inactive until all nodes had started the transmission of their fifth message. The connectivity matrices are changed every second. Please note that the absolute values of the time units are not important for the simulation, only their relation (1:10). They could also have been set to 6 seconds and one minute, yielding the same results.

Figure 2 shows the OMNeT++ representation of our simulations. An array of nodes of size *numNodes* represents the sensor network. The *connectionManager* enables changing connections between nodes, while the switches from one connectivity matrix to the next are realized by the *matrixSwitch*-module. Additionally, a global logging component has been added, which records (among others) the number of successfully delivered messages.

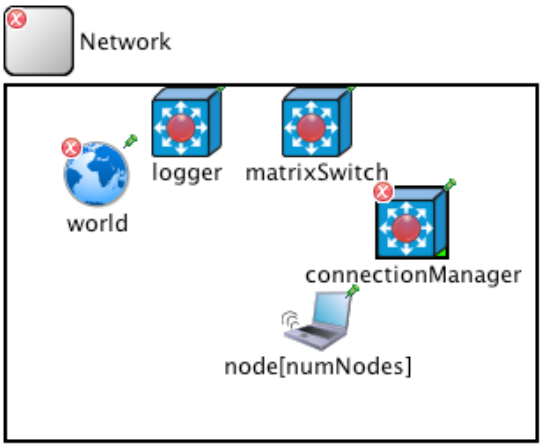


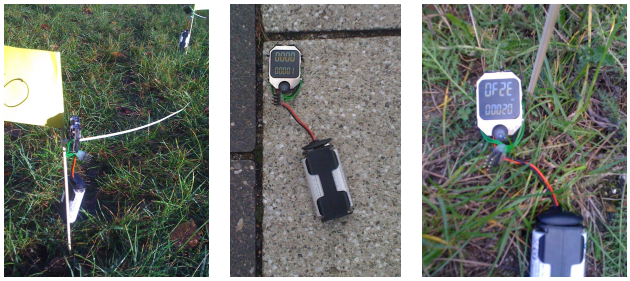
Figure 2: Sensor Network Simulation Components

In the real experiments, each node wants to transmit a message every minute. The experiments run for one hour each, therefore, 60 messages are transmitted by the application on each node. In all experiments, 36 nodes are placed in a square of six times six. Each node records the number of application messages it received and all nodes record the number, type, and size of all messages they transmit or forward. The sensor nodes were placed in four different locations: on a desk, affixed to poles, placed on a lawn, and placed onto a stone pavement. The transmission power was set to 0dBm.

For all real world experiments, eZ430-Chronos Sensor nodes from Texas Instruments [4] were used. The eZ430-Chronos is an inexpensive evaluation platform for the CC430. It features an MSP430 micro controller with an integrated CC1100 sub-gigahertz (868MHz) communication module [1]. The evaluation board is delivered as a compact sports watch containing several sensors, e.g., a three-axis accelerometer and five buttons which are connected through general purpose I/O pins. The sports watch casing has been removed in order to use the eZ430s as sensor nodes.

Figure 3 shows the used eZ430-Chronos sensor nodes. An external battery pack has been soldered to the nodes, and replaces the internal coin cells. This enables the usage of freshly charged batteries for each experiment.

Apart from the modification for the batteries, the sensor nodes were used as they were delivered, no calibration was made. This should reflect the fact that future users would neither be able nor willing to calibrate a large number of nodes. Instead, they are used "out of the box". The transmission power was also left at the preset level of 0 dBm, which lead to a small transmission range. This small transmission range is also due to the absence of a real antenna on the eZ430-Chronos: The metal surrounding the display acts as the antenna.



(a) affixed to poles (b) on a stone pavement (c) placed on the lawn

Figure 3: A modified eZ430-Chronos Sensor Node

Desk Experiments.

This deployment is a single hop layout, where each node is able to receive messages from every other node. The nodes lay directly next to each other. An old set of batteries was used without re-charging them, because range did not really matter in these experiments. They were used to validate the correct operation of the protocols.

Poles.

For the pole experiments, small poles were deployed on the lawn in front of the main building of our university, with about one meter distance between each of them. Then, the sensor nodes were affixed to them using cable straps, at a height of about 20 cm (Figure 3(a)). The pole placement was usually used at 8am.

Lawn.

After the pole experiments were finished and evaluated, the nodes were reset and placed on the ground directly next to the poles as shown on Figure 3(c). The resets were done by disconnecting the batteries and reconnecting them directly afterwards. The same set of batteries as before was used on each node without charging. The lawn experiments started at about 10 AM.

Stones.

After the lawn experiments, the nodes were disconnected and poles as well as nodes and batteries collected. The experiments on the stones always started at about 1 PM, using the same set of 72 AA batteries used in the morning without re-charging, but the pairing of batteries and nodes might have changed, i.e., the batteries that were connected to node 4 in the pole and lawn experiments might be connected, e.g., to node 27 in the stone experiments. These experiments were conducted on the stone pavement on our campus (Figure 3(b)).

3.1 Comparison between Simulations and Experiments

There are nearly no limits to network size in the simulations. However, the comparison should be made between networks of the same size and diameter. As the real world experiments were conducted with 36 nodes, networks also consisted of 36 nodes in the simulations. The matrices were generated with ten different seed values for the random number generator. Also, three different values for link probability were used. Moreover, the destination was switched between nodes. This resulted in 1080 simulations for each protocol. In the real world experiments, the placement on the desk represented a single hop environment and the poles only provided two hops distance. Therefore, these two sets of experiment locations are dismissed for the evaluation.

We evaluated the performance of four routing protocols: *Dynamic Source Routing* (*DSR*) [7], *Tree Routing*, *Flooding* and *Buckshot Routing* [14].

DSR has been chosen because it was one of the first routing protocols which took unidirectional links into account. It offers two modes of operation: One for only bidirectional links and one for unidirectional links. In both modes, Route Request (RREQ) messages are flooded through the network to find the initial route from source to destination. Once a RREQ has reached the destination, a route reply message (RREP) is transmitted. If only bidirectional links are present, the RREP takes the same route as the RREQ and the route is discovery is finished when this RREP reaches the source. When unidirectional links are considered, the route taken by the RREQ might contain unidirectional links and not be usable in the reverse direction. Therefore, the RREP, which contains the route from source to destination, is also flooded. When the first RREP message arrives at the source, the path from source to destination is known. Then, the destination still needs to be informed about the route from destination to source, which the RREP collected on its way to the source. This path is then transmitted in a single, not flooded message, as the path from source to destination is known.

Tree Routing still remains the most used routing protocol in wireless sensor networks. In tree routing, the sink transmits a tree building message. All nodes that receive this message record the sink as their parent node and transmit a tree building message of their own. All nodes that receive such a message and do not yet have a parent node follow the same principle, until a routing tree has been built. In our implementation, we used tree routing with two retransmissions per hop.

Flooding is the most simple routing protocol, it only needs a duplicate suppression algorithm. No routing tables are filled or maintained. A node that wants to transmit a message to the sink simply transmits it. Every node that receives this message checks its duplicate suppression mechanism. If the node has already forwarded the message before, it is discarded. Otherwise it is retransmitted.

Buckshot Routing has been specifically designed for wireless networks with unidirectional links. It is based on a multi path approach which implicitly uses unidirectional links. When a node received a message, it normally checks whether it is the intended next hop. If it is not, the message is discarded in other protocols. In Buckshot Routing, the node does not check if it is the intended next hop. Rather, it checks its neighbor table to find out if it is a neighbor of the next but one hop (the hop after the next). This way, not only the intended next hop but also all its siblings forward the message, leading to a spread around the original route. This spread represents redundancy, which enables the implicit usage of unidirectional links, supplies a circumvention around failed nodes and ignores temporarily broken links. However, this redundancy has a price: Buckshot Routing transmits more messages than traditional routing protocols. These transmission cost are justified by the increased delivery ratio.

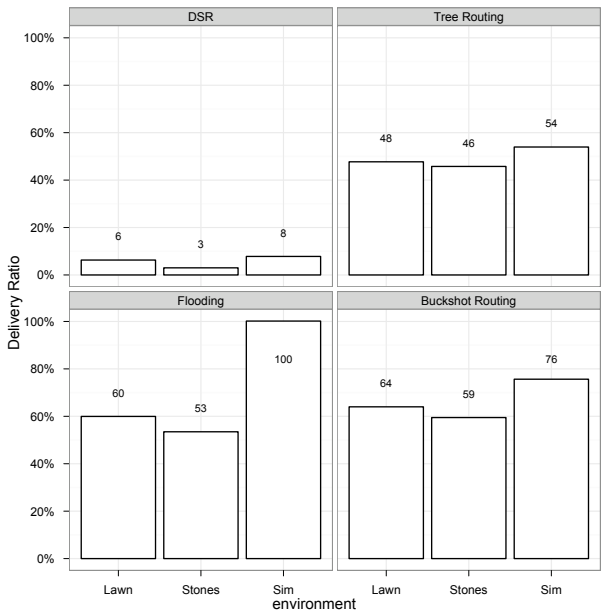


Figure 4: Delivery Ratio of each Protocol; Experiments vs. Simulations

Figure 4 shows the median of the delivery ratio for all evaluated protocols for the two multihop (4-5 hops) experiments (lawn, stones) and the median from 1080 simulations for each protocol. Naturally, the results of *Flooding* in the simulation are much better than those achieved in the real world experiments, as *Flooding* suffers heavily from the broadcast storm problem in the real experiments. The CSMA MAC layer provided by the hardware simply cannot handle the huge number of messages. Apart from *Flooding*, the simulation results and those of the two experiment settings are quite similar and minor differences can be explained with irregularities during the experiments.

For *DSR*, which can operate with unidirectional links, the frequency of link changes presented a huge problem. While it is able to work perfectly with static unidirectional links, link changes and route breaks result in a large number of messages transmitted during route repair. Therefore, *DSR* is a prime example for the reason why simulation models that only use different transmission strengths, i.e., static links, are inadequate.

For *Tree Routing*, the small network diameter and the 2 retransmissions on each hop were sufficient to deliver about 50% of application messages. When the network diameter grows (simulations not shown here), it runs into huge performance problems.

For *Buckshot Routing*, which has been explicitly designed for networks with unidirectional links, the network size was too small. The main advantages, implicit route repair and multiple routes, only make a real impact in larger networks (simulations not shown here).

In summary it can be said that the used simulation approach has some limitations, as it does not include the medium access control protocol used in the real experiments. However, the results show that the usage of connectivity matrices and the way they were generated is close to reality, and can be used to evaluate the influence of unidirectional links and frequent link changes on routing protocols. This is exactly what the simulations were intended for as the used MAC layer and other side effects of the used hardware might (and hopefully will) change for future deployments.

Another advantage of the developed simulation model is the fact that connectivity data gathered during connectivity measurement experiments can easily be included. When a network deployment is planned, it is possible to place the nodes in the field and measure the connectivity. Transforming the measured data into connectivity matrices can be done automatically. These matrices can then be used as input for our simulation model, which can in turn be used to evaluate the proposed communication protocols before the actual deployment.

4. RELATED WORK

"A Link-Layer Tunneling Mechanism for Unidirectional Links" [3] has been proposed by the unidirectional link routing group (UDLR) [13] at the Internet Engineering Task Force (IETF) [5]. Its main goal is to make unidirectional links usable in the internet. There, unidirectional links have a different nature then those dealt with in this paper. As RFC 3077 deals with internet connections, the links are stable, and unidirectional links exist over a really long period. An example for unidirectional links as mentioned in RFC 3077 are satellite connections, where the satellites can transmit to a lot of receivers ("local" broadcast), but the receivers cannot transmit back to the satellite.

One assumption made by the authors of the RFC is that nodes can be divided into three categories: `Receivers`, `Send-only feed` and `Receive capable feed`. `Receivers` are on the lower end of a unidirectional link, i.e., have an incoming only link. `Send-only feeds`, e.g., satellites, have an outgoing unidirectional link. `Receive-capable feed` are routers that have "send-and-receive connectivity to a unidirectional link" [3].

Another assumption made is that each router has more than one IP connection, allowing for the tunneling of messages.

The basic idea behind the tunneling mechanism is the forwarding of link layer messages of one interface using the routing layer of another interface on the same node when this link is unusable.

This approach offers the possibility of using any routing protocol over the tunneling mechanisms, and hides the existence of unidirectional links from them. It cannot, however, hide the longer delay, which can be a huge problem for timeouts used in the routing protocols. Also, as stated by the authors, this tunneling mechanism does not work "where a pair of nodes are connected by 2 unidirectional links in opposite direction" (using different interfaces). This refers to the fact that all links on the tunnel have to be bidirectional. If the link from node B to node C in the example above was unidirectional, the mechanism described in RFC 3077 would have failed, even though a detour existed.

The authors of [12] propose a combined evaluation method that uses experiments with real hardware, emulation and simulation techniques in order to speed up the deployment of new protocols. The combination of all three methods enables the developer to identify problems and shows where further investigation is necessary. The routing protocols *AODV*, *DSR* and *OLSR* were used to evaluate the proposed

approach to protocol monitoring. They found that latency and timing are crucial to the performance of reactive protocols like *AODV* and *DSR*, because of buffering times. The queue-ups that can result from this buffering were apparent in their experiments, but not in the emulations.

In conclusion of this paper, it can be said that all three methods of evaluation have their own advantages for a protocol developer, if they are used correctly. For simulations, the choice of the underlying communication model is crucial. The emulation can be fed with real world connectivity data and can be used to evaluate the implications of the network stack used on the real devices. Experiments are needed to generate this connectivity data. It is important that for all three methods exactly the same implementation of the protocol is used, and that this implementation is the one that can be used directly on the hardware which is used in the real experiments.

REFLEX [20] is an operating system for deeply embedded systems and sensor nodes that has been developed by the distributed systems/operating systems group at the Brandenburg University of Technology Cottbus, Germany. It is based on the event flow principle which removes the need for explicit synchronization within components.

REFLEX is implemented in C++, which enables the application programmer to use state of the art object oriented programming methods. Also, this fact enables REFLEX to be used on a range of different platforms, because all that is needed to deploy it is a C++ compiler and a few lines of assembler code for hardware specific drivers. To enable its use in wireless sensor networks, a power management scheme has been integrated [17] and is continuously being improved. All protocols are implemented operating system independent, but an operating system has to be used nonetheless. For the reasons listed above, REFLEX has been chosen as operating system for the real world experiments and simulations.

OMNeT++ [19] is a discrete event simulator that can be used to simulate different kinds of networks. OMNeT supplies a framework of modules which can be combined to form compound modules.

Both types of modules contain gates, which can be connected using channels, to allow the modules to communicate with each other. This is done by passing messages from one module to the other. OMNeT is implemented in C++, which made the integration of REFLEX into OMNeT possible [9].

MiXiM [8] is a simulation framework for OMNeT++. It provides an abstraction for communication layers, namely MANET and sensor network communication. Explicit simulation of unidirectional links using a connectivity matrix has been added to MiXiM in our simulation model.

5. CONCLUSION

In this paper we presented a matrix based simulation approach for wireless sensor networks with unidirectional links and frequent link changes. Links are represented by connectivity matrices and replaced with new ones frequently. We evaluated our simulation model by comparison of simulation and experiment results for selected routing protocols. In the future we hope to obtain more connectivity data from real network deployments which can replace the randomly generated matrices.

6. REFERENCES

[1] Texas instruments cc430f6137, `http://focus.ti.com/docs/prod/folders/print/cc430f6137.html`.

[2] B. B. Chen, S. Hao, M. Zhang, M. C. Chan, and A. L. Ananda. Deal: discover and exploit asymmetric links in dense wireless sensor networks. In *Proceedings of the 6th Annual IEEE communications society conference on Sensor, Mesh and Ad Hoc Communications and Networks*, SECON'09, pages 297–305, Piscataway, NJ, USA, 2009. IEEE Press.

[3] E. Duros, W. Dabbous, H. Izumiyama, N. Fujii, and Y. Zhang. A link-layer tunneling mechanism for unidirectional links, http://www.faqs.org/rfcs/rfc3077.html, Mar 2001.

[4] Texas instruments ez430-chronos, `http://focus.ti.com/docs/toolsw/folders/print/ez430-chronos.html?DCMP=Chronos\&HQS=Other+OT+chronos`.

[5] I. E. T. Force. http://www.ietf.org/.

[6] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Trans. Sen. Netw.*, 2(1):1–38, 2006.

[7] D. Johnson, D. Maltz, and J. Broch. *DSR The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[8] A. Koepke, M. Swigulski, K. Wessel, D. Willkomm, P. Klein Haneveld, T. Parker, O. Visser, H. Lichte, and S. Valentin. Simulating wireless and mobile networks in OMNeT++: The MiXiM vision. In *1st Int. Workshop on OMNeT++*, mar 2008.

[9] A. Lagemann and J. Nolte. Integration of event-driven embedded operating systems into omnet++ – a case study with reflex. In *2nd International Workshop on OMNeT++*, Rome, Italy, March 2009.

[10] S. Lohs, R. Karnapke, and J. Nolte. Link stability in a wireless sensor network - an experimental study. In *3rd International Conference on Sensor Systems and Software*, 2012.

[11] M. K. Marina and S. R. Das. Routing performance in the presence of unidirectional links in multihop wireless networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '02, pages 12–23, New York, NY, USA, 2002. ACM.

[12] E. Nordström, P. Gunningberg, C. Rohner, and O. Wibling. Evaluating wireless multi-hop networks using a combination of simulation, emulation, and real world experiments. In *MobiEval '07: Proceedings of the 1st international workshop on System evaluation for mobile platforms*, pages 29–34, New York, NY, USA, 2007. ACM.

[13] U. L. R. G. of the IETF. http://www.udcast.com/udlr/.

[14] D. Peters, R. Karnapke, and J. Nolte. Buckshot routing - a robust source routing protocol for dense ad-hoc networks. In *Ad Hoc Networks Conference 2009*, Niagara Falls, Canada, 2009.

[15] L. Sang, A. Arora, and H. Zhang. On exploiting asymmetric wireless links via one-way estimation. In *MobiHoc '07: Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, pages 11–21, New York, NY, USA, 2007. ACM Press.

[16] L. Sang, A. Arora, and H. Zhang. On link asymmetry and one-way estimation in wireless sensor networks. *ACM Trans. Sen. Netw.*, 6(2):12:1–12:25, Mar. 2010.

[17] A. Sieber, K. Walther, S. Nürnberger, and J. Nolte˙Implicit sleep mode determination in power management of event-driven deeply embedded systems. In *7th International Conference on Wired / Wireless Internet Communications*, University of Twente, The Netherlands, 2009.

[18] Turau, Renner, and Venzke. The heathland experiment: Results and experiences. In *Proceedings of the REALWSN'05 Workshop on Real-World Wireless Sensor Networks.*, Jun 2005.

[19] A. Varga. The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic, June 2001.

[20] K. Walther, R. Karnapke, and J. Nolte. An existing complete house control system based on the reflex operating system: Implementation and experiences over a period of 4 years. In *Proceedings of 13th IEEE Conference on Emerging Technologies and Factory Automation*, 2008.

[21] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 14–27, New York, NY, USA, 2003. ACM Press.

[22] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 1–13, New York, NY, USA, 2003. ACM Press.