Let's Move: Adding Arbitrary Mobility to WSN Testbeds

Nils Aschenbruck*•, Jan Bauer°, Jakob Bieling°, Alexander Bothe°, and Matthias Schwamborn*
*University of Osnabrück - Institute of Computer Science
Albrechtstr. 28, 49076 Osnabrück, Germany
{aschenbruck, schwamborn}@informatik.uos.de
°University of Bonn - Institute of Computer Science 4
•Fraunhofer FKIE
Friedrich-Ebert-Allee 144, 53113 Bonn, Germany
{bauer, bieling, bothea}@cs.uni-bonn.de

***Abstract*—Research in the area of Wireless Sensor Networks (WSNs) has been immense during the last years. Since highly resource-constrained, WSNs pose specific challenges to the design and implementation of protocols and algorithms. In order to test, compare, and verify the intended functionality of new approaches, performance evaluations must be conducted in a sound and credible manner. Instead of simulative evaluation, the WSN research community has mostly shown a preference towards experimental evaluation. Being especially important for the evaluation of routing protocols, node mobility is one of the more complex features to facilitate in testbeds. Robots have been the most common means for moving nodes so far, which, however, introduces more costs and limits reproducibility, scalability, and mobility patterns. We present a new software-based approach that essentially combines mobility modeling with link control. Mobility patterns taken from the mobility scenario generator BonnMotion are converted and replayed to create a virtual dynamic topology. Our approach drastically reduces costs, makes mobility reproducible and scalable, and enables the use of a variety of mobility models.**

***Index Terms*—Mobile communication; Performance evaluation; Wireless sensor networks**

I. Introduction

Wireless Sensor Networks (WSNs) are known as wireless multi-hop networks composed of small sensor devices that monitor changes of some measurable phenomenon, e.g., light, temperature, or movement. Applications of WSNs range from military (e.g., security perimeter surveillance) over civilian (e.g., disaster area monitoring) to industrial (e.g., industrial process control). In contrast to notebooks and smartphones, embedded sensor nodes (*motes*) are highly constrained by processing power, memory, battery power, and network capacity. Therefore, specific challenges arise in the design and implementation of protocols and algorithms (for a comprehensive overview, see [2], [22]): Code size and energy-efficiency are two of the major concerns in WSNs since program memory is limited to only a few kilobytes and most WSN deployments are designed for maximum network life-time.

Since Mobile Ad hoc NETwork (MANET) protocols were not designed with the above-mentioned requirements in mind, their use in WSNs is quite limited. Therefore, the research community has focused on the development of new protocols at all layers tailored to meet these requirements. In order to test, compare, and verify the intended functionality of these new approaches, performance evaluations need to be conducted in a sound and credible manner.

Even though testbed research and development was conducted by the MANET community (cf. [8], [13]), simulation has been chosen most commonly for performance evaluation. In the WSN research community, however, experimental evaluation with real motes is preferred. One of the more complex features to integrate into a testbed is node mobility, which is especially important for the evaluation of routing protocols. The need for cost reduction, reproducibility, and realism makes it even more challenging. Robot movement has been the most prominent approach to integrate mobility into WSN testbeds so far (cf. Section II). This leads to extra costs, extra hardware maintenance, and movement is limited by the capabilities of the robots. Further limitations concern reproducibility and scalability.

We propose a new software-based approach that essentially combines mobility modeling with link control. The physical testbed remains static, i.e., no physical movement of nodes is involved. Instead, the idea is to apply the mobility-induced dynamic topology with dynamic link control. Mobility patterns are taken from the mobility scenario generation and analysis tool BonnMotion [3], which means that every implemented mobility model can be utilized. Our approach drastically reduces costs, makes mobility reproducible and scalable, and enables the use of a variety of mobility models.

The rest of this paper is structured as follows. First, we discuss related work found in the literature on mobility in testbeds in Section II. Then, a detailed description of our virtual mobility approach in Section III follows. In Section IV, we present some evaluation results achieved with our implementation. Finally, we conclude the paper and point out directions for future work in Section V.

II. Related Work

Integrating mobility into testbeds has been a challenging issue at least since the emergence of the first MANET testbeds [8]. In the Ad hoc Protocol Evaluation (APE)

TABLE I
SURVEY OF WSN TESTBEDS CONCERNING MOBILITY INTEGRATION.

Testbed	**Application**	**Mote**	**Mobility**	
BWN-Lab [1]	multimedia communication	MicaZ	Acroname Garcia robot	**real mobility**
Explorebots [6]	general purpose (indoor)	Mica2	Rogue ATV robot	
Robomote [7]	general purpose (indoor)	Mica2	Robomote robot	
Kansei [9]	large-scale sensing	TelosB Trio XSM	Acroname robot	
Mobile Emulab [12]	general purpose (indoor)	Mica2	Acroname Garcia robot	
TWiNS.KOM [19]	smart heterogeneous networking	SunSPOT TelosB	moving object or person	

testbed [14], [15], laptops are carried around by people within a specified campus environment. Mobility scenarios are realized through choreographed testruns, i.e., each person carrying a laptop gets individual message popups indicating their movement behavior (e.g., "Stay at this location. (30 sec)."). Another approach to testbed mobility is pursued by Netbed [20], where passive couriers (busses and students) carry around mobile devices. For these approaches, scalability is a challenge since an appropriate number of node carriers is required.

In [13], mobility integration is categorized into three different abstraction levels:

- *Real mobility*: This is the lowest level of abstraction. Physical node positions are changed by manual movement (human carriers), by automatic movement (robots), or by switching between antennas of fixed location. Appropriate regulation of radio transmission powers might also be involved, otherwise enough space for multi-hop topologies is needed. Common drawbacks are lack of scalability and reproducibility.
- *Channel emulation*: This is a medium level of abstraction. Testbed nodes are stationary and radio signals are artificially altered to reflect the signal propagation properties of a radio channel according to the considered mobility scenario. This usually involves usage of special hardware components like switches, multiplexers, or attenuators. Channel emulation can be considered as a trade-off between realism and reproducibility.
- *Logical connectivity*: The highest level of abstraction is achieved by reducing the impact of mobility on wireless links to logical connectivity. The dynamic topology can be described by a time-variant connectivity matrix, which is the result of applying a signal propagation model to the mobility scenario description. This makes mobility much more controllable and reproducible. However, the reduction to logical connectivity also involves the loss of accuracy in link dynamics.

Note that even though this classification is based on MANET testbeds, it is generic enough to be applied to WSNs.

In WSN testbeds, mobility is mostly realized by robotic movement: Table I contains a survey of different WSN testbeds with mobility support, including application scenario, deployed mote platforms and mobility concept. BWN-Lab [1], Explorebots [6], Robomote [7], Kansei [9], and Mobile Emulab [12] realize mobility by robotic movement, where the Acroname Garcia platform seems to be a popular choice. Reinhardt et al. [19] also mention robotic movement as a possible approach for their TWiNS.KOM testbed, but did not implement it. Instead, they rely on moving objects or persons. According to the classification by Kropff et al. [13], as described above, these six testbeds use the *real mobility* abstraction level.

Robotic movement might in fact be a good choice if the user wants to control node movement during an experiment and still obtain reproducible results to a certain degree. Another benefit is that robots can move in reaction to certain events, which cannot always be pre-computed. However, robots also mean additional costs, additional hardware maintenance, and movement is limited by their capabilities. Furthermore, reproducibility and scalability are also limited and the overhead to program and control them should not be neglected.

Also related to our work is the Virtual Mobility Overlay (ViMobiO) for WSN testbeds by Puccinelli et al. [18]. Based on an idea first mentioned in the context of the ORBIT testbed for MANETs [17], they propose an overlay scheme that enables virtual mobility by transferring state information of logical nodes between physical nodes. The positions of the physical nodes predefine the waypoints of the logical nodes. Virtual movement of a logical node l from waypoint w_1 to w_2 is equal to transferring the state of l from physical node p_1 to p_2, where w_1 and w_2 are the positions of p_1 and p_2, respectively. Since state information as well as regular data packets are sent over the wireless channel, time is divided into epochs consisting of a period for regular traffic and a period for state traffic to clearly separate one from the other. Moving state of l from p_1 to p_2 is only possible if

(1) there is no other logical node occupying p_2 and
(2) p_2 is listed in the neighbor table of p_1.

(1) has several implications: First, it means that there may be no more than one logical node occupying a physical node at a time. Second, the number of logical nodes must be significantly smaller than the number of physical nodes. Otherwise, logical nodes would restrict each other's movement or even deadlock states might occur. Finally, the mobility patterns of nodes are not independent of each other. Requirement (2) further restricts movement since logical nodes cannot move to a physical node multiple hops away without visiting the hops in between.

Another drawback of ViMobiO is the interruption of regular code execution during the state transfer phase of each epoch. Since there usually is no interruption of application code in a

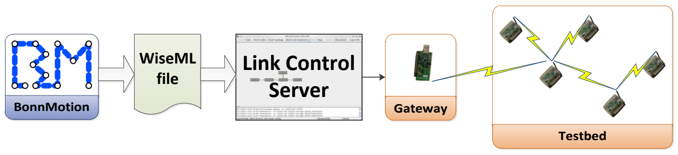


Fig. 1. Virtual mobility concept.

real network deployment, this should have significant impact on the experiment's results. Nevertheless, the authors neither explain the handling of this code interruption nor present any results on its impact.

Our approach to virtual mobility differs fundamentally from ViMobiO. Instead of having a network overlay with a subset of nodes, our approach can be considered as using an overlay with a different set of links. Moreover, it supports arbitrary mobility patterns and does not interrupt the execution of application code.

III. Virtual Mobility

We propose a novel virtual mobility approach based on link control to enable arbitrary mobility patterns in a static WSN testbed. Speaking in terms of the aforementioned mobility classification [13], with our approach, we apply the *logical connectivity* abstraction level. The basic idea is to generate a dynamic topology description with a mobility scenario generator tool and use this as input for the link control. The link control then enables or disables links during an experiment according to the description. This way, virtual movement can be achieved while the nodes remain physically static. Overall, this approach has several benefits:

(1) No extra costs for additional hardware (incl. maintenance),
(2) no need for node carriers,
(3) reproducibility,
(4) scalability, and
(5) utilization of commonly used and realistic mobility models.

The different parts of our virtual mobility concept are depicted in Figure 1. The first step is to use the mobility scenario generation and analysis tool BonnMotion [3], [5] for generating a mobility scenario in form of a dynamic topology description (Section III-A). The generated description is then available in form of a WiseML-compliant file. This file includes the dynamic topology description which can be interpreted by the Link Control Server (LCS), the central coordinator for link control (Section III-B). Via serial connection, the LCS sends link change commands to a mote gateway. Finally, this gateway forwards the commands over the wireless channel to the WSN testbed.

A. BonnMotion

BonnMotion is an open-source Java software tool for the generation and analysis of mobility scenarios. It supports multiple random-based as well as scenario-specific mobility models (for a complete list, see [5]). Node movement is natively stored in the form of *waypoints*. A waypoint can be defined as a triple $(id, time, pos)$, where

- id is the node ID,
- $time$ is the (simulation) time relative to the beginning of the scenario, and
- pos is the position of the node within the specified simulation area.

The complete trajectory of a mobile node can be determined by connecting every two chronologically subsequent waypoints of the same node with a straight line.

The waypoint description of a mobility scenario is frequently used for network simulations. For the link control concept, however, we need a dynamic topology description, i.e., information about link establishment and link break. Therefore, we compute so-called *contacts* (commonly considered in the field of opportunistic networks), which denote encounters of two nodes with respect to some predefined communication range. A link between two nodes n_1 and n_2 is considered as *established* as soon as n_2 enters the communication range of n_1. Likewise, it is considered as *broken* as soon as n_2 leaves the communication range of n_1. The time difference between link establishment and link break is then called *contact duration*. All nodes share the same communication range, resulting in symmetric links. Link asymmetry is supported by WiseML as well as by the LCS, but we consider link asymmetry and (being related) more complex signal propagation models as out of scope for this paper.

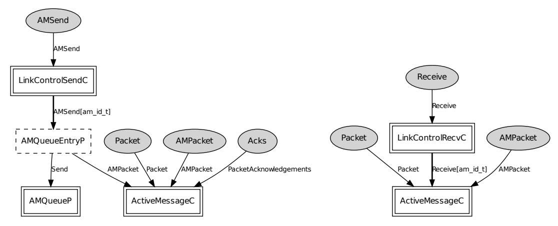
WiseML [21] is a scenario and experiment specification language for WSNs that is based on GraphML, an XML dialect. It is also used by Baumgartner et al. for creating virtual testbed federations with virtual links [4]. The topology edges are defined by `enableLink`/`disableLink` XML elements, both with `source` and `target` attributes for unidirectional links support.

In order to specify the dynamic topologies resulting from mobility scenarios generated with BonnMotion, we extended the WiseML exporter of BonnMotion version 1.5a to compute contacts and support contact-based output. The contact-based extension reports the link status every time there is a change in the topology according to the user-defined transmission range. At the beginning of the mobility scenario, links are disabled or enabled depending on the initial topology. Whenever there is a link change between two nodes n_1 and n_2, the following information is written to the WiseML file:

- Time of the link change event,
- position of both nodes at this time within the simulation area,
- corresponding XML element for link enabling or disabling, once for each link direction ($n_1 \rightarrow n_2$ and $n_2 \rightarrow n_1$).

At the end of the mobility scenario, all links are disabled such that all communication is stopped.

Converting the waypoint-based format to the contact-based format is basically a reduction of complete node trajectories to



(a) DirectAMSenderC (b) AMReceiverC

Fig. 2. Integration of link control layer into TinyOS base code.

the impact of node mobility on the wireless links. This is accompanied by loss of information on fine-grained movement, i.e., loss of accuracy in link dynamics. However, this is an inherent drawback of the logical connectivity abstraction level and is traded in for more control, reproducibility, and ease of use.

B. Link Control

We wrote a TinyOS [11] link control implementation for enabling and disabling physical links. In order to obtain arbitrary virtual topologies with this approach, we assume a fully connected testbed. We don't consider arbitrary physical topologies of the testbed since they are hard to create and maintain due to fading effects and might change over time. The implementation consists of three parts (cf. Figure 1): Server (LCS), gateway, and clients (testbed), i.e., the actual network to run experiments on. The LCS is a Java program responsible for reading and replaying the WiseML-formatted mobility scenario input by disseminating the corresponding commands. The gateway mote is the interface between clients and server. It passes packets it receives from the server via serial connection to the clients via wireless radio and vice versa. We used the `BaseStation` application available in the TinyOS distribution since it already provides this functionality. The clients receive, process, and acknowledge the commands sent by the server.

An experiment using virtual mobility works as follows. Initially, all link changes in the WiseML file containing the mobility scenario are scheduled according to their timestamps by the LCS. When a link change event concerning $n_1 \rightarrow n_2$ is due, it is transmitted to the gateway via serial connection (UART) in the form of two `enablePhysicalLink` or two `disablePhysicalLink` commands, respectively. The unidirectional link change results in two commands (one for n_1 and one for n_2) since every client maintains one list for each link direction. However, since we consider symmetric links only here, we combined incoming and outgoing link change commands (with the same timestamp): Instead of sending a command for $n_1 \rightarrow n_2$ and $n_1 \leftarrow n_2$ separately, we only send a command for $n_1 \leftrightarrow n_2$. The gateway then sends these control packets to the corresponding nodes (n_1 and n_2) via single-hop on the wireless channel. Especially the initial virtual topology dissemination introduces a lot of control packets. In order to avoid potential traffic congestion, the server applies a minimum pause time between the transmission of two consecutive packets. As soon as a client node receives a link control command, it edits its *whitelist* for incoming or outgoing links, respectively. A response is sent back to the server for confirmation. We opted for wireless transmission of control data mainly for two reasons: Not having to rely on wired connections makes the testbed much more flexible. The other reason is that not all mote platforms are equipped with a USB connector. Some need an extra gateway module for this which increases cost of the testbed. Note, however, that this also introduces potential interference with regular data traffic.

As already mentioned, each client maintains two whitelists: One for incoming links and one for outgoing links. Both whitelists are initially empty, such that only broadcast messages can be sent and control packets from the gateway can be received. Node addresses are added and deleted during the experiment, depending on the link control command. If a unicast data packet is to be sent to a receiver which is not listed in the outgoing links whitelist, the most significant bit of the destination address is set (we assume that this bit is 0 in regular addresses). The modified packet is transmitted but the original destination never reached. On one hand, this ensures that the disabled link to the receiver is in fact not used. On the other hand, it still causes interference which is also caused by real node movement: A real mobile node does not abruptly stop the retransmission of a packet if it moves out of the receiver's transmission range. If a packet (unicast or broadcast) is received from a source node which is not listed in the incoming links whitelist, it is dropped. An exception to this rule are link control packets, which are always processed.

The main part of the code is implemented as a library. Based on the TinyOS wiring concept, the link control layer can be easily switched on and off with a compiler flag. From the TinyOS (v2.1.1) base code, we only had to modify `DirectAMSenderC` and `AMReceiverC` (see Figure 2). We configured `DirectAMSenderC` to use the link control sender module (`LinkControlSendC`) as a filter layer for outgoing packets (cf. Figure 2a). Likewise, we configured `AMReceiverC` to use the link control receiver module (`LinkControlRecvC`) as a filter layer for incoming packets (cf. Figure 2b). This wiring ensures that the link control layer is transparent to higher layers. Furthermore, since no hardware-specific code was touched, our implementation works for all hardware platforms supported by TinyOS.

IV. Evaluation

We experimentally evaluated our virtual mobility approach in our WSN testbed. The goal of the evaluation is to examine

- if higher layers are affected by the link control layer,
- if mobility can be reproduced,
- how large the control packet overhead is, and
- how large the link control transmission delay is.

We will first explain the evaluation scenario setup and then present and discuss the results.

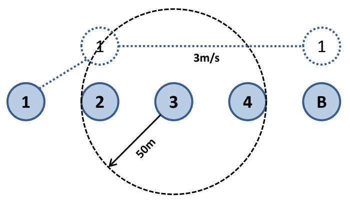


Fig. 3. Mobility scenario used in the evaluation.

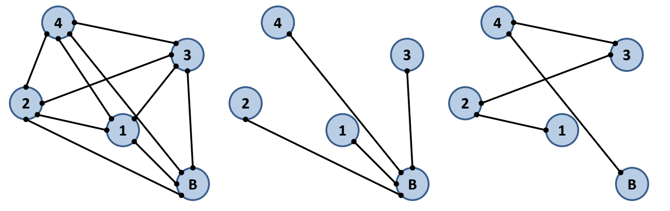


Fig. 4. Comparison of different topology contexts in the evaluation scenario.

A. Setup

Starting with the mobility scenario, we chose a chain topology with one mobile node (cf. Figure 3). We opted for this simple scenario since it makes it easier to comprehend the impact of mobility on the network. It consists of four regular nodes (1-4) and a base station (B). As shown in Figure 3, they are aligned as a chain with order $1, 2, 3, 4, B$. After an initial wait phase of 20 s, node 1 begins to move along the chain towards node B as indicated by the dotted lines. When it has reached the waypoint position above node B (indicated by the small dotted circle), it moves back along the same way to its original position. The speed of node 1 is 3 m/s and the distance between two neighboring nodes is 50 m. Node 1 returns to its original position at 157.3 s. The mobility scenario ends at 180 s such that there are more than 20 s at the end where the topology is static again. We convert the waypoint-based format of the scenario to contact-based WiseML and choose a transmission range of 50 m. This leads to the desired chain topology, where a wireless link only exists between direct neighbors.

We use the TelosB mote platform to run our experiments on. The positioning of the five nodes in our testbed lab looks similar to the one in Figure 4. Since the nodes are close to each other, the network is fully connected, i.e., wireless links exist between each pair of nodes. Therefore, the *physical topology* looks like Figure 4a. All nodes run TinyOS v2.1.1 with a simple sensor data collect application called `WSNCollect`. It uses the tree-based Collection Tree Protocol (CTP) [10] for routing. Node B functions as the root node, while node 1 transmits sensor data packets with a sampling interval of 1 s. Nodes 2-4 simply forward packets towards B and do not generate own traffic. Using `WSNCollect` without the virtual mobility scenario would result in a 1-hop tree structure due to CTP routing and high quality links. This *logical topology* is shown in Figure 4b. Applying the virtual mobility scenario yields the desired chain-like *virtual topology* in Figure 4c.

TABLE II
CODE SIZE OF LINK CONTROL CLIENT IMPLEMENTATION ON TELOSB MOTES.

Application	**ROM**	**RAM**
`WSNCollect` (incl. Link Control)	24 948 B	3638 B
`WSNCollect`	23 862 B	3046 B
Link Control only	**1086 B**	**592 B**

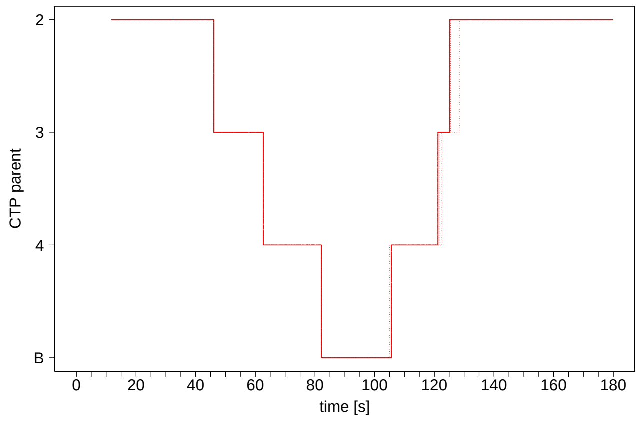
In addition to the five nodes, we used another TelosB mote running the TinyOS `BaseStation` application (gateway node) and a laptop running the LCS. A Jackdaw IEEE 802.15.4 sniffer enabled us to capture the packets transmitted over the air using the packet analyzer Wireshark (for details, see [16]). The gateway as well as the sniffer were placed near node B.

During first experiment runs, we noticed that the Trickle timer as part of CTP does not work well with dynamic topologies. It is responsible for timing the transmission of CTP routing beacons. These beacons are required for neighbor discovery and routing table updates. Since the Trickle timer basically has exponential growth and a default maximum value of 512 s, neighbor discovery can be slow. If, in our mobility scenario (cf. Figure 3), node 1 moves within transmission range of, e.g., node 3, it may well be out of range again before the Trickle timer of node 3 fires. The result is that node 1 misses the chance to discover node 3 and choose it as its new parent. Instead, it still tries to send its data packets to node 2, which is, however, long out of range. To solve this problem, we disabled the Trickle timer and implemented a simple periodic timer with a period of 2 s instead.

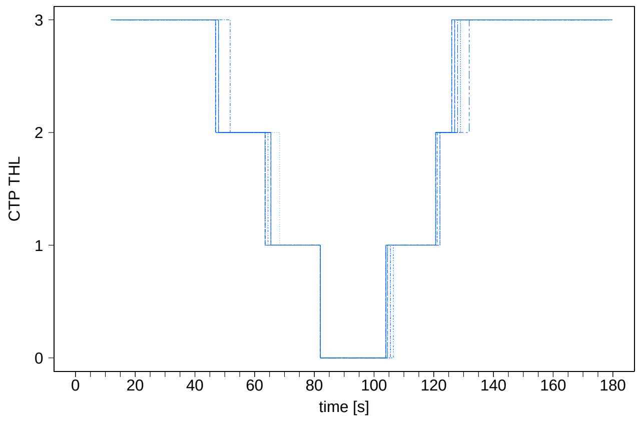
B. Results

As mentioned earlier, the link control layer can be switched on and off using a compiler flag. We compiled the `WSNCollect` application with and without the layer to get an exact value of the code size. Table II shows the resulting code sizes in terms of ROM and RAM on the TelosB platform. Our link control implementation has a *small code footprint* of 1086 B in ROM and 592 B in RAM. RAM usage can be further decreased by defining a smaller whitelist size in the code (default is 64 links per whitelist).

For the evaluation, we ran 10 replications of the scenario described in the previous subsection. All of the results were calculated by parsing the Wireshark traces and processing timestamps of the LCS. As a first metric, we consider the "parent" field in the CTP routing packets. The parent of a node is the next hop towards the root. Therefore, we expect mobile node 1 to change its parent accordingly as it moves towards root node B. The step plot in Figure 5a shows the parent of node 1 at a specific time as broadcasted in its periodic routing packets. As expected, the parent changes from node 2 to 3, 4, and finally B as the node moves along the chain. It changes again in reverse order as node 1 moves back to its original position. Replications have been plotted with different line types and can be made out as displaced vertical lines at parent change times. These displacements indicate that the



(a) Parent



(b) THL

Fig. 5. CTP packet field entries verifying the intended mobility pattern.

parent change times vary between replications, which is mostly due to the varying wireless channel.

Closely related to the parent field in the context of the considered mobility scenario, the CTP Time Has Lived (THL) field denotes the number of hops a data packet has traversed so far. We examined the THL field for each data packet received by root node B. Since node 1 is the only node sending data packets, the THL values should coincide with the parent field entries. Therefore, we also show the THL with a step plot (see Figure 5b). As expected, the number of hops decreases from 3 (parent node 2) to 0 (parent node B) as node 1 moves towards B. It increases again as the mobile node moves away from the root node towards the other end of the chain. Again, we can distinguish between replications by displacements at THL change times.

The evaluation of both considered CTP fields shows that the mobility pattern was carried out as intended. Thus, the link control layer is transparent to higher layers and these layers work as expected. Furthermore, the mobility scenario can be reproduced within the bounds of wireless channel fluctuation affecting the control communication.

Next, we evaluate the control packet overhead introduced by the LCS. For this purpose, we calculated the sum of control packet sizes within intervals of 5 s for each replication. The resulting packet overhead in bytes per 5 s is shown in Figure 6a. Replications have been aggregated to mean and standard deviation (error bars). As expected, the initial virtual topology dissemination (up to 10 s) introduces a lot of control communication. However, since the initial phase of a network performance evaluation is usually cut off anyway (also done in Figure 5), the initial overhead can be neglected. After the initial dissemination, the overhead does not increase any further until node 1 moves into transmission range of node 3. From there on, the control overhead increases on a regular basis every time a new link change occurs according to the movement along the chain. The overhead decreases as soon as the mobile node returns to its original position. Overall, apart from the initial phase, the control overhead is well below 80 B/5s (16 B/s), which is negligible considering the IEEE 802.15.4 data rate of 250 kbps.

As a final metric, we consider the delay of the control communication in form of the Round-Trip Time (RTT), i.e., the time between submission of a link control command and reception of the corresponding response. We calculated the RTT on application level for each link change in all replications and cut off samples corresponding to the initial phase of 10 s. The remaining samples result in the Empirical Cumulative Distribution Function (ECDF) shown in Figure 6b. The RTT values vary between 15.02 ms and 38.78 ms with a mean of 23.36 ms. This order of magnitude for the delay is negligible for most mobility scenarios since the impact of even lower inter link change times for a single node on routing performance is insignificant.

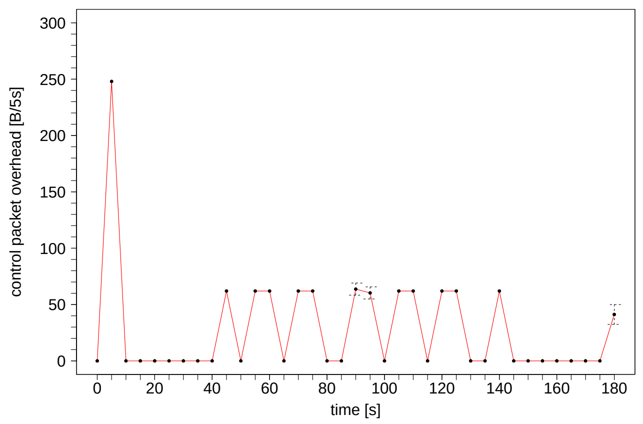
V. Conclusion

We presented a new software-based approach to enable arbitrary virtual mobility in a static WSN testbed. Mobility modeling and link control are combined to create a mobility-induced dynamic topology. We have shown that our approach leaves a small code footprint, makes mobility reproducible, and the control communication introduces negligible overhead in terms of size and delay. Moreover, it drastically reduces costs, makes mobility scalable, and enables the use of a variety of mobility models.

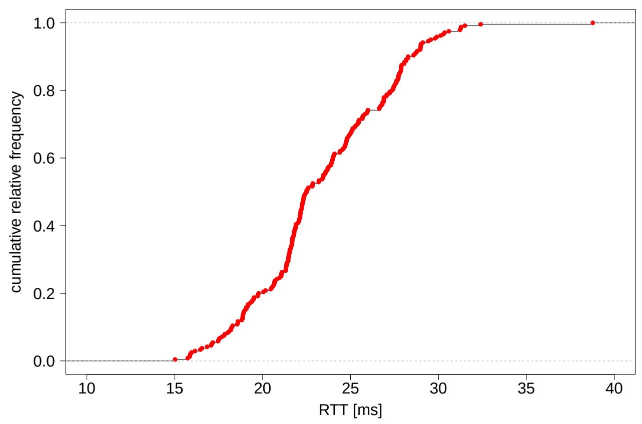
In the future, we want to improve some aspects of the virtual mobility. E.g., the use of more realistic signal propagation models should increase accuracy of link dynamics. Furthermore, we are planning to evaluate various protocols and algorithms in different mobility scenarios.

Acknowledgments

This work was supported in part by the German Federal Office for Information Security (BSI). Furthermore, this work was supported in part by CONET, the Cooperating Objects



(a) Overhead



(b) RTT

Fig. 6. LCS control packet overhead and RTT.

Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2-224053. The authors would like to thank the WSNLab and CONET project teams for feedback, sustainable discussion, and work.

References

[1] I. Akyildiz, T. Melodia, and K. Chowdhury, "Wireless Multimedia Sensor Networks: Applications and Testbeds," *Proc. IEEE*, vol. 96, no. 10, pp. 1588–1605, 2008.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[3] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, and M. Schwamborn, "BonnMotion - A Mobility Scenario Generation and Analysis Tool," in *Proc. of the 3rd Int. Conference on Simulation Tools and Techniques (SIMUTools '10)*, Torremolinos, Malaga, Spain, 2010, pp. 1–10.

[4] T. Baumgartner, I. Chatzigiannakis, M. Danckwardt, C. Koninis, A. Kröller, G. Mylonas, D. Pfisterer, and B. Porter, "Virtualising Testbeds to Support Large-Scale Reconfigurable Experimental Facilities," in *Proc. of the 7th European Conference on Wireless Sensor Networks (EWSN '10)*, Coimbra, Portugal, 2010, pp. 210–223.

[5] BonnMotion Developers, "BonnMotion - A Mobility Scenario Generation and Analysis Tool," 2011. [Online]. Available: http://bonnmotion.net.cs.uni-bonn.de/

[6] T. Dahlberg, A. Nasipuri, and C. Taylor, "Explorebots: A Mobile Network Experimentation Testbed," in *Proc. of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '05)*, Philadelphia, PA, USA, 2005, pp. 76–81.

[7] K. Dantu, M. Rahimi, H. Shah, S. Babel, A. Dhariwal, and G. Sukhatme, "Robomote: Enabling Mobility in Sensor Networks," in *Proc. of the 4th Int. Conference on Information Processing in Sensor Networks (IPSN '05)*, Los Angeles, CA, USA, 2005, pp. 404–409.

[8] P. De, A. Raniwala, S. Sharma, and T. Chiueh, "Design Considerations for a Multihop Wireless Network Testbed," *IEEE Commun. Mag.*, vol. 43, no. 10, pp. 102–109, 2005.

[9] E. Ertin, A. Arora, R. Ramnath, M. Nesterenko, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, and H. Cao, "Kansei: A Testbed for Sensing at Scale," in *Proc. of the 5th Int. Conference on Information Processing in Sensor Networks (IPSN/SPOTS '06)*, Nashville, TN, USA, 2006, pp. 399–406.

[10] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection Tree Protocol," in *Proc. of the 7th Int. Conference on Embedded Networked Sensor Systems (SenSys '09)*, Berkeley, CA, USA, 2009, pp. 1–14.

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Networked Sensors," *SIGPLAN Not.*, vol. 35, pp. 93–104, November 2000.

[12] D. Johnson, T. Stack, R. Fish, D. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile Emulab: A Robotic Wireless and Sensor Network Testbed," in *Proc. of the 25th Conference on Computer Communications (INFOCOM '06)*, Barcelona, Spain, 2006, pp. 1–12.

[13] M. Kropff, T. Krop, M. Hollick, P. Mogre, and R. Steinmetz, "A Survey on Real World and Emulation Testbeds for Mobile Ad hoc Networks," in *Proc. of the 2nd Int. Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM '06)*, Barcelona, Spain, 2006, pp. 448–453.

[14] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordström, and C. Tschudin, "A Large-scale Testbed for Reproducible Ad hoc Protocol Evaluations," in *Proc. of the Wireless Communications and Networking Conference (WCNC '02)*, Orlando, FL, USA, 2002, pp. 412–418.

[15] E. Nordström, P. Gunningberg, and H. Lundgren, "A Testbed and Methodology for Experimental Evaluation of Wireless Mobile Ad hoc Networks," in *Proc. of the 1st Int. Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM '05)*, Trento, Italy, 2005, pp. 100–109.

[16] C. O'Flynn, "RZRAVEN USB Stick (Jackdaw)," 2011. [Online]. Available: http://www.sics.se/~adam/contiki/docs-uipv6/a01108.html

[17] M. Ott, I. Seskar, R. Siraccusa, and M. Singh, "ORBIT Testbed Software Architecture: Supporting Experiments as a Service," in *Proc. of the 1st Int. Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM '05)*, Trento, Italy, 2005, pp. 136–145.

[18] D. Puccinelli and S. Giordano, "ViMobiO: Virtual Mobility Overlay for Static Sensor Network Testbeds," in *Proc. of the 10th Int. Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM '09)*, Kos, Greece, 2009, pp. 1–6.

[19] A. Reinhardt, M. Kropff, M. Hollick, and R. Steinmetz, "Designing a Sensor Network Testbed for Smart Heterogeneous Applications," in *Proc. of the 33rd Conference on Local Computer Networks (LCN '08)*, Montreal, Quebec, Canada, 2008, pp. 715–722.

[20] B. White, J. Lepreau, and S. Guruprasad, "Lowering the Barrier to Wireless and Mobile Experimentation," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 47–52, 2003.

[21] WISEBED Project Partners, "WiseML Schema Version 2.1," 2011. [Online]. Available: http://dutigw.st.ewi.tudelft.nl/wiseml/wiseml_schema.pdf

[22] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless Sensor Network Survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.