Electronic Communications of the EASST
Volume 37 (2011)

Workshops der wissenschaftlichen Konferenz
Kommunikation in verteilten Systemen 2011
(WowKiVS 2011)

Towards Virtual Mobility Support in a Federated Testbed for Wireless Sensor Networks

Torsten Braun, Geoff Coulson and Thomas Staub

12 pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Towards Virtual Mobility Support in a Federated Testbed for Wireless Sensor Networks

Torsten Braun[1], **Geoff Coulson**[2] and **Thomas Staub**[1]

[1] Institute of Computer Science and Applied Mathematics
University of Bern
Neubrückstrasse 10, CH-3012 Bern, Switzerland
braun|staub@iam.unibe.ch

[2] School of Computing and Communications
Lancaster University
Lancaster LA1 4WA, England
geoff@comp.lancs.ac.uk

Abstract: We present a design that accommodates 'virtual mobility' in a wireless sensor network testbed. Virtually-mobile nodes can be physical, simulated or emulated, and virtual mobility for all three types is treated uniformly by embedding the nodes in a virtual space. In operation, the traffic of virtually-mobile nodes is intercepted and redirected to a mobility model from where it is selectively forwarded to other nodes that are virtually in range. We present a distributed implementation architecture that potentially allows the simulation/emulation of large-scale wireless sensor networks with large numbers of virtually-mobile nodes.

Keywords: Mobility, testbed, wireless sensor networks, emulation, simulation

1 Introduction

Research on wireless sensor networks (WSNs) has grown rapidly in recent years, and large-scale experimental deployments of WSNs are now becoming widespread [DHK+09]. This rapid growth has led to a strong emerging requirement for *flexible experimentation facilities* to support the design and evaluation of new protocols and mechanisms for WSNs. To this end, a number of testbeds for WSNs have been built, among them WISEBED (www.wisebed-eu) [CKM+09] which forms the background of this paper. WISEBED is an extremely flexible federated testbed that supports experimentation at all levels from applications down to low-level communications. It addresses the need for flexibility through the concept of *virtual testbeds*, in which physical, simulated, and emulated testbed elements can be freely mixed.

In WISEBED *virtual mobility* has been identified as an area that needs further development. In response, we present a design for virtual mobility that can be implemented in WISEBED and potentially in other testbed environments. The remainder of the paper is structured as follows. Section 2 surveys related work and in Section 3 we discuss the concept of virtual mobility in detail and consider general issues in the design space of virtual mobility infrastructure. In Section 4 we describe our design of a virtual mobility infrastructure. Finally, in Section 5 we offer our conclusions and discuss future work.

2 Related Work

Various attempts have been made to integrate virtual mobility into testbeds, although to our knowledge these have all been targeted at wireless networks in general rather than WSNs in particular. In addition, current efforts typically do not support a scalable federated testbed environment nor the co-existence of virtual and physical mobility. The approach in [ESHF04] integrates virtual mobility with real network stacks and operating systems by virtualizing host connectivity via the *MobiEmu* 802.11b network emulator [ZL02]. One drawback of MobiEmu is that it does not model errors: i.e. communication is either possible without errors or not at all. JiST/MobNet [KBHS07] provides a Java framework for the simulation, emulation and real-world testing of wireless ad hoc networks as an wireless extension of the *Java in Simulation Time* (JiST) simulator. As communication software is not usually written in Java, it cannot be tested directly. The *Emulab* [WLS⁺02] testbed, which was originally developed for wired networks, has been extended to wireless networks [WLG02] by means of an IEEE 802.11a/b/g testbed. However, besides its lack of mobility support, Emulab suffers from limited repeatability due to interference from other networks in a shared office building. The *ns-3* network simulator [LWD⁺09] allows the integration of virtualized nodes running native applications and protocol stacks under the Linux operating system. Virtualized nodes in ns-3 are connected through a TUN/TAP kernel device and a proxy node to the simulation. However, there is no support to modify (wireless) device parameters of the simulation directly and dynamically by the virtualized nodes. In order to cope with the problem of simulator overload during network emulation, a central synchronizer has been proposed in [WSHW08] to control the time flow of virtual hosts using an adapted scheduler for XEN, keeping them synchronized using the *OMNeT++* network simulator [Var01]. Another approach is emulation based on hardware channel simulators [JS03, BJSS09, WVBC09, SBBD03]. The main advantage of such approaches is repeatability of experiments in combination with realistic media access control (MAC) and physical layers. The main drawbacks are the high costs and the limited number of supported nodes.

In our own related work, VirtualMesh [SGB11] provides instruments to test real communication software in a controlled environment. VirtualMesh intercepts and redirects real traffic generated by real nodes to an OMNeT++ simulation model, which then calculates packet transmission according to the virtual network topology, propagation model, background interference and current node positions. Only the MAC and physical layers are simulated; the other layers remain unchanged and work as in a real testbed of embedded Linux nodes. VirtualMesh has proven to be scalable, to have minimal influence on throughput and to introduce only negligible delays (less than 0.4 ms per hop). However, this work does not support specifically WSN environments, federated testbeds, or the co-existence of virtual and physical mobility.

3 Virtual Testbeds and Virtual Mobility

3.1 Virtual Testbeds

WISEBED's 'virtual testbed' (henceforth VTB) concept offers the abstraction of a dedicated, private WSN testbed in which some of the 'primary testbed elements' (see below) are physically real, some are simulated and others are emulated [CKM⁺09]. The user designs a VTB in such a way that its mix of physical, simulated, and emulated elements is appropriate to their goals. They then instantiate their VTB in the federated WISEBED environment, deploy their software onto it, and observe the behaviour and outputs of their experiment as if it were running on a local physical testbed. The full list of 'primary testbed elements' that can be virtualised in a VTB is as follows: *sensors* (e.g., temperature devices); *sensor input* (i.e., what the sensors observe, such as the current temperature); *nodes* (the microcontroller + memory + radio device to which sensor devices are attached); *power* to nodes; *connectivity* (which is a function of node location and radio characteristics); and, finally, *mobility* of nodes. To date, WISEBED has explored virtualisation of all of these elements with the exception of virtual mobility.

So-called *virtual links* are an important element in the underpinning of the VTB concept, which we also use in our virtual mobility design. Virtual links are used to define potential connectivity between pairs of nodes that are virtually close but physically distant (e.g. located at different sites in the WISEBED federation). Specifying a virtual link represents the possibility of 1-hop unidirectional communication between two nodes in the VTB. The specification of a virtual link includes its radio characteristics such as the packet error rate [BCD^{+}10]. Within individual nodes (be they physical, simulated or emulated), WISEBED's *driver stacking framework* [BCF^{+}10] is used to engineer the end points of virtual links. This framework is used to deploy 'pseudo' radio drivers that appear to software on the node as 'real' radio drivers; however these pseudo-drivers transparently divert (selected) outgoing packets to the virtual link machinery, and insert incoming packets arriving from the virtual link machinery.

3.2 Virtual Mobility

In the context of a VTB we define *virtual mobility* as follows: a VTB features virtual mobility if, during the course of a user's experiment on the VTB, the positions of nodes (be they physical, simulated, or emulated), as initially specified in the VTB description, change. To support virtual mobility in VTBs, we propose the following basic concept: Where we have a VTB featuring virtual mobility, all communication between virtually mobile nodes and other nodes (whether virtually mobile or fixed) is channelled through a *virtual mobility interpreter* (VMI) that maintains a list of the current (virtual) locations of all the nodes involved. All packets sent from nodes are channeled to the VMI using components from the virtual link machinery mentioned above. When the VMI receives a packet it pushes it through a radio channel model, and thereby discovers a virtual area within which the packet can be heard. The VMI then pushes the radio packet to all nodes within this virtual area. Again, this pushing is done through the virtual link machinery. Drilling down into this basic design it is clear that there are many issues arising:

1. How do we *specify* virtual mobility? One option is to use WISEBED's XML-based WiseML language, which has most of the required descriptive capability. But there are other options such as the 'BonnMotion format' [AEGS10] (http://www.cs.uni-bonn.de/IV/BonnMotion) which specifies a plain text file in which every line describes the motion of one host[1]. Another option would be to employ vector-based descriptions of mobility – these would not need to rely on interpolation between successive discrete instances of

[1] In this format, a line consists of one or more (t,x,y) triplets of real numbers, such as `t1 x1 y1 t2 x2 y2 t3 x3 y3 t4 x4 y4` ...; the meaning being that the given node will get to location (x_k,y_k) at time t_k.

time. And a more left-field option would be to adopt a virtual time concept as proposed by [WSHW08]. A related design dimension is whether a mobility specification is fixed or changeable:

(a) The mobility of a node may be *predefined* prior to an experiment, e.g., as part of the experiment initialization using, e.g., the BonnMotion format.

(b) Or, mobility may be *dynamically calculated* during the simulation by the supervising VMI or the node itself.

2. How does the system *control* virtual mobility? The obvious approach is to let a controlling entity such as the VMI specify the nodes' mobility. We call this a *controlled mobility* specification, since a controlling entity rather than the individual node itself determines its mobility specification. Another possibility is *autonomous mobility* whereby individual nodes determine their own virtual mobility – e.g. according to private scripts owned by each node. In this case, the individual nodes would need to continually update the VMI with their current position. The option of autonomous mobility would also facilitate the modelling of adaptive mobility – or at least it would delegate to individual nodes the problem of how to adapt mobility.

3. How could the VMI concept *be realised*? One possible approach would be to use a specialised hardware-based engine to model high resolution, high throughput radio channels. However, this approach is likely to be expensive and to not scale very well. Another approach, which we currently favour, is to use a simulator engine such as OMNeT++ as the basis of the VMI. As mentioned in Section 2, experience has indicated that real-time throughput can be achieved with this simulator at a reasonable scale [SGB11]. Furthermore, it is obvious that scaling could be further enhanced through distribution—especially given that WISEBED is already a federated architecture. For example, we could have one OMNeT++-based VMI instance at each physical testbed site (they could be time synchronised using NTP or PTP) or assign VMIs to sub-areas of the virtual space. This raises further questions of how to optimally place VMIs within the physical infrastructure underlying the VTB to maximise scalability. It would clearly be important in a distributed VMI implementation to minimise communication latency between VMI instances.

4. Can virtual mobility *co-exist with physical mobility*? Ideally, any physically mobile nodes in a VTB environment should work consistently with virtual mobility elsewhere in the VTB – so that, for example, if two physical nodes are within range of each other, and one moves physically and the other virtually, but in the same direction and at the same speed, the (virtual) distance and therefore (virtual) connectivity between the two stays the same. A related issue is: when we have virtual mobility in a VTB do we force *all* communication through the VMI - or can we leave some nodes in the VTB to communicate using their native mechanisms (such as physical radios, virtual links) outside of the VMI context – e.g. if some nodes are out of communication range of anything that might move. But given that nodes are moving how does the system knows which areas of the VTB are 'safe' from virtual mobility? This can be deduced in fixed and deterministic mobility scenarios, but presumably not when nodes can autonomously virtually move.

4 Implementation Approach for the Virtual Mobility Concept

Having considered the issues above we now present the design of an infrastructure to support virtual mobility in a WISEBED VTB environment. The main elements in our design are as follows (see Figure 1):

- A set of nodes (physical, emulated or simulated) that comprise the VTB.
- A *virtual space*, divided into *sub-spaces*, in which nodes live and in which virtual mobility takes place.
- A distributed set of VMIs, each of which is associated with a sub-space of virtual space.
- A distributed set of *portal server agents* (PSAs). A PSA runs on the portal server of each physical testbed site.[2] Simulator servers that host simulated nodes will also have an associated PSA.
- An *initiator* that is responsible for initialising the system.

We proceed by first considering a 'basic case' design and then, in the following section, considering complicating factors and extensions.

4.1 Basic Case

4.1.1 Preamble

Each virtually-mobile node has a dynamically-varying coordinate attribute (x,y,z), which places it somewhere in virtual space. The virtual space is divided up between VMIs in such a way that the sub-spaces associated with each VMI slightly overlap such that each VMI can see nodes just over the 'border'. We refer to these overlapping areas as *interference areas*; and to the other areas, where no overlaps exist, as *core areas*. The extent of the interference areas is determined such that packet transmissions from nodes within an interference area *might* affect nodes in the neighbour sub-spaces, while transmissions originating from a core area will definitely not affect any neighbour sub-space.

The motivation for this interference/core distinction is to handle cases where packets travel over borders, and the VMIs on each side of the border need to be aware of all in-range nodes so that they can factor them into their modelling of radio interference.

At any given time, we will say that nodes within a VMI's core area are *supervised by* that VMI. Where a node is in an interference area, we say that the node is supervised by one and only one of the VMIs participating in the interference area, and is additionally *co-supervised by* the others. We will also say that each node is *managed by* some specific PSA at the physical implementation level. The 'supervised by' and 'co-supervised' mappings might change as the node virtually moves; but the 'managed by' mapping will be invariant as it is part of the physical infrastructure (barring physical mobility).

[2] In the WISEBED federated architecture a portal server is a gateway interconnecting a testbed site to the Internet.

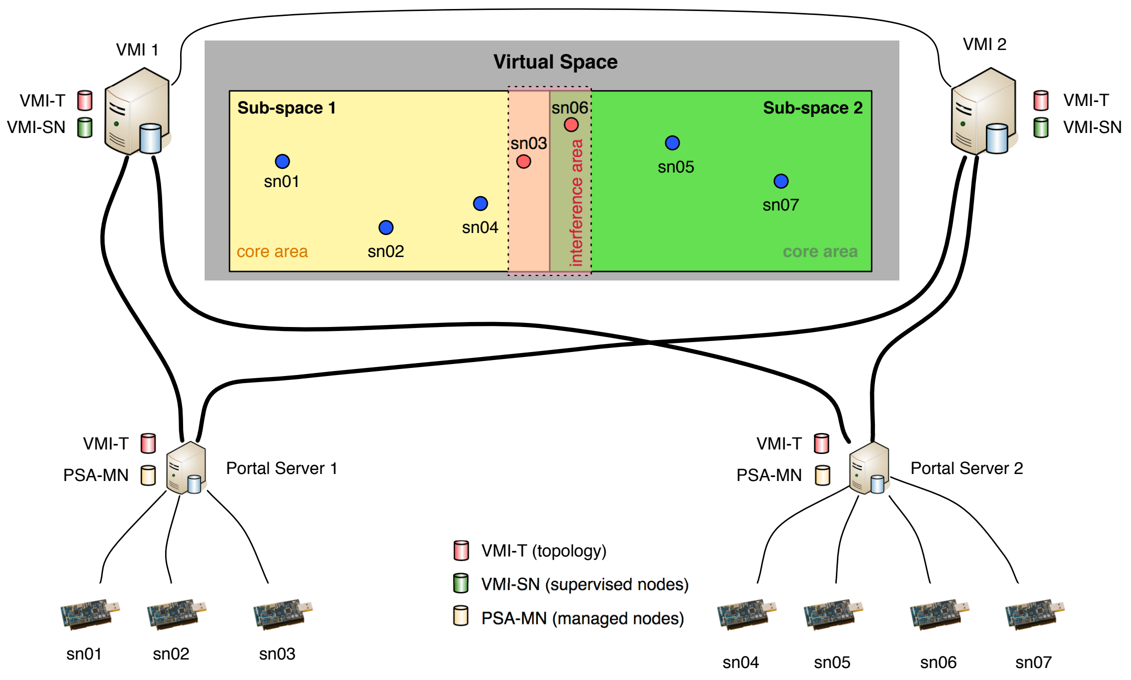

Figure 1: General architecture showing the main components with two PSAs and two VMIs

4.1.2 Data Structures

The design incorporates the following data structures.

- Each VMI has a *supervised node table* (VMI-SN) that maps the nodes it is supervising to the node's current coordinate attribute value and to its managing PSA.
- Each VMI has a *VMI topology table* (VMI-T) that maps each VMI to coordinates in the virtual space. This table is the same for each VMI. Each PSA also has a copy of this table.
- Each PSA has a *managed node table* (PSA-MN) that maps its managed nodes to a VMI.

The VMIs are interconnected using an overlay network which is structured according to the relative positions of the VMIs' sub-spaces. For the construction of the overlay network we employ Voronoi diagrams, which map each point in virtual space to the closest Voronoi site (i.e. the point's associated VMI). The Delaunay triangulation corresponding to the Voronoi diagram defines the VMI overlay network (see Figure 2). This ensures low latency connectivity between neighbouring VMIs because neighbouring VMIs are directly interconnected in the overlay topology (this is advantageous since neighbouring VMIs communicate frequently with each other).

4.1.3 Initialization

Initialization of a virtual-mobility enabled VTB is performed by the initiator, which sets up the experiment by defining the virtual sub-spaces and their associated VMIs. The overlay network

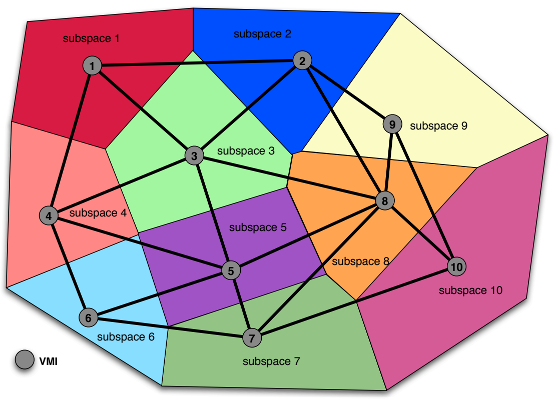

Figure 2: VMI overlay network based on Delaunay triangulation and a Voronoi diagram

discussed above is used to disseminate information about the virtual positions of the VMIs so that they can initialize their VMI-T tables[3]. The PSAs initialize their copies of the VMI-T table by contacting one or more VMIs.

At the beginning of an experiment the nodes and their managing PSAs have to be defined. Moreover, mobility information must be provided, which is stored in the VMI-SN tables (in addition, a backup copy is kept at the managing PSA in order to be able to recover from VMI failures). In case of pre-defined mobility (see Section 3) this mobility information includes full trajectories of the nodes from which VMIs can initialize their VMI-SN tables. In the case of dynamically-calculated mobility, it contains mobility-model dependent parameters based (e.g. these parameters would relate to the OMNeT++ framework assuming our mobility model builds on the VirtualMesh approach discussed in Section 2).

4.1.4 Virtual Mobility Management

To keep the PSA-MN and VMI-SN tables up to date, dynamic information such as the nodes' supervising VMIs is disseminated by flooding or by direct communication between a VMI and the PSAs that manage its supervised nodes.

To support virtual mobility across sub-spaces, a *handover mechanism* between VMIs has to be implemented. Handovers are triggered by the previously-supervising VMI as soon as a node is deemed sufficiently close to a neighbouring sub-space. In the pre-defined mobility case a VMI should have enough knowledge (current position, speed, direction, and future mobility pattern) to determine the best time for triggering a handover. To minimize the number of handovers, it is not necessary to trigger a handover as soon as a node is is seen to be approaching a neighbour VMI. It is also foreseen that VMIs can keep track of nodes that have recently left the interference area and are in a core area of a neighbouring VMI. In case of a handover, the VMI-SN tables of the VMIs involved in the handover will change. Moreover, the new supervising VMI will inform

[3] This information can also regularly disseminated throughout the overlay network as in link-state routing protocols. Note, however, that topology information only needs to be updated if the virtual position of VMIs can change, as discussed in Section 4.2.3.

the corresponding PSA about that change. The mobility information of a node is also forwarded to the new VMI accordingly.

4.1.5 Packet Processing

Having sketched how virtual mobility works, we are now in a position to describe in detail what happens when a virtually-mobile node sends a packet:

1. The packet is intercepted by the WISEBED's driver stacking framework on the sending node and forwarded to the node's PSA.
2. The PSA looks up the sending node's currently supervising VMI.
3. The PSA forwards the packet to the supervising VMI.
4. The supervising VMI determines all neighbouring sub-spaces affected by the packet transmission (this is done using a rough model that over-estimates packet propagation) and forwards a copy of the packet to the asociated VMIs[4], together with all relevant transmission parameters such as the position of the transmitting node and the transmission power used.
5. All the VMIs involved push the packet through their radio models (e.g. as provided by OMNeT++) ; as a result, they determine the actual, accurate, spatial extent to which the packet should propagate.
6. All the VMIs consult their VMI-SN tables to determine the set of supervised (but not co-supervised) nodes that currently lie within this extent and thus the PSAs that are managing each of these nodes.
7. All the VMIs that supervise receiving nodes forward packets to the PSAs managing those nodes.
8. The receiving PSAs forward the packet to the respective receiver nodes.

We now exemplify this protocol with specific reference to Figure 3. Note that nodes *sn*03 and *sn*06 are in the interference areas of their respective sub-spaces: *sn*03 is supervised by VMI 1 and co-supervised by VMI 2; whereas *sn*06 is supervised by VMI 2 and co-supervised by VMI 1 (the step numbers in the below correspond to those above).

1. Node *sn*03 sends a packet which is intercepted and transferred to PSA 1
2. PSA 1 consults its VMI-T table to determine *sn*03's supervising VMI (i.e. VMI 1).
3. PSA 1 forwards the packet to VMI 1.

[4] This requires that neighbour VMIs can communicate with a rather short delay. Alternatively, the PSA could send packets to all VMIs that might be affected by the upcoming transmission, e.g., send the packets to all neighbours of the supervising VMI. This information is available at the PSAs' copy of the VMI-T table. However, processing such a transmission would require the neighbour VMI to know exactly the transmitting node's position. This can be achieved by frequently exchanging node position information between VMIs, in particular about nodes in their common interference area.

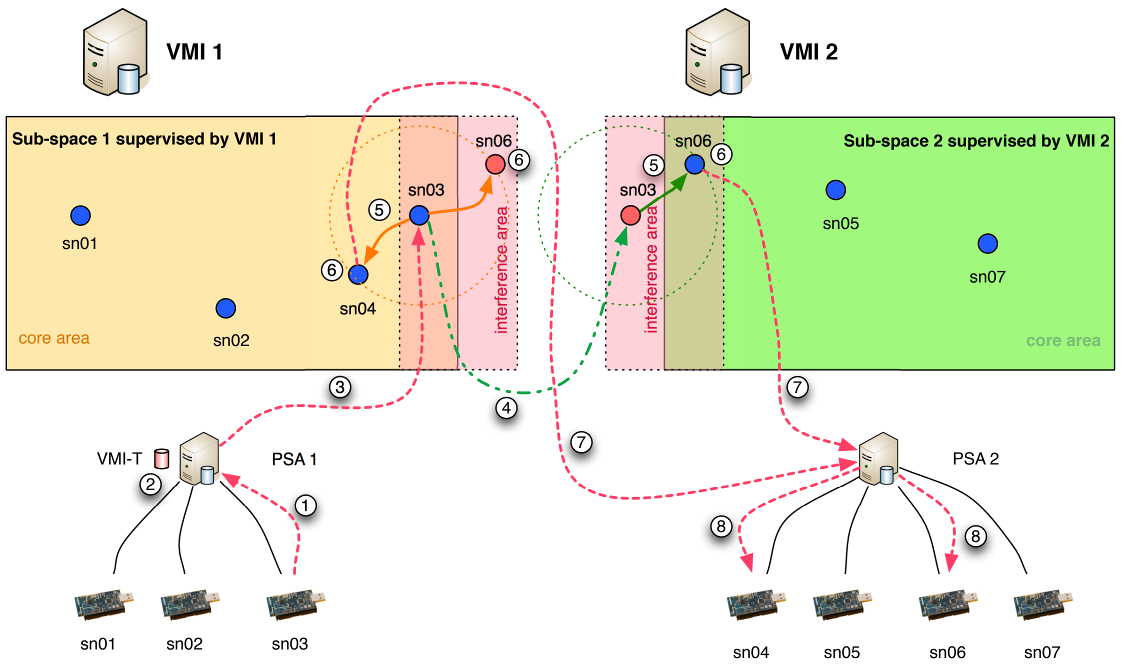

Figure 3: Example packet flow: sn03 sends to sn04 and sn06

4. VMI 1 forwards the packet to VMI 2, since *sn*03 is in their common interference area.
5. The propagation of the packet is modelled by both VMIs using their configured propagation model and taking interference into account.
6. As a result, VMI 1 determines that the packet should be received by *sn*04 and, in parallel, VMI 2 determines that the packet should be received by *sn*06.
7. VMI 1 forwards the packet to PSA 2 (destination: *sn*04); and VMI 2 forwards the packet to PSA 2 (destination: *sn*06)
8. PSA 2 forwards the packets to *sn*04 and *sn*06 respectively.

4.2 Discussion of More Complex Cases and Further Extensions

4.2.1 Autonomously Mobile Nodes

In the basic design discussed above, we assumed that virtual mobility was controlled by a mobility model at the VMI or another controlling entity. We now briefly consider the case in which each node autonomously determines its mobility, meaning that nodes must proactively inform PSAs / VMIs about their virtual positions.

In this case, each node knows its virtual position and can pass this information to the managing PSA with each packet transmission. The PSA can then directly forward the packet to all relevant VMIs, i.e., to the supervising VMI as well as to the neighbour VMIs. Basically, this uses the alternative direct packet forwarding from the PSA to potentially affected VMIs that was discussed in step 4 above. Furthermore, we have to ensure that an autonomously mobile node periodically updates its position via the PSA to its supervising VMI in order to support correct processing of received transmissions.

4.2.2 Accommodating Physical Mobility

Physical mobility[5] can be accommodated by applying a design similar to that discussed in Section 4.2.1, in which autonomous nodes (including physically mobile ones) repeatedly report their positions to PSAs and VMIs. In this case, the reporting process would have to apply a mapping from physical floor coordinates to the virtual space's coordinate system. This mapping, which would be established statically at the start of the experiment, would correspond to exactly where in the virtual space the floor space (and thus the limits of the node's mobility) was supposed to be located. When a physically-mobile node sends a message, it would be handled by the usual PSA/VMI machinery and would therefore have exactly the same semantics as the virtually-mobile case. The same would apply for the case of a physically-mobile node receiving packets from virtually-mobile nodes. The same would even apply for one physically-mobile node communicating with another physically-mobile node.

4.2.3 Adaptation of Sub-Spaces

Until now we have assumed a fixed partitioning of the virtual space into sub-spaces. A further extension is to let the VMIs self-organize and adapt the sizes of their virtual sub-spaces. This might be useful for automatic load balancing. Before starting the experiment, the initiator would do the initial definition of the VMIs' virtual positions. Afterwards, the overlay network and the respective Voronoi diagram would be adaptive in order to support load balancing. For example, a VMI might move closer to another VMI in order to cover more nodes. Another criterion for VMI movement might be to reduce node handovers. Frequent handovers might occur if a densely-populated area with high node mobility is divided by a sub-space border. In any case, node movement should not violate the planar graph characteristic. Information about the virtual positions of the VMIs as well as their number of supervised nodes would be regularly disseminated throughout the overlay network as in link-state routing protocols.

5 Conclusions and Next Steps

We have presented an architecture for adding virtual mobility to a federated testbed for wireless sensor networks. Our concept adds virtual mobility by embedding physical, emulated and simulated nodes in a common virtual space in a completely uniform manner. The traffic generated by nodes is intercepted and redirected to a mobility model (VMI) responsible for managing a virtual space. Our concept provides unlimited mobility of all types of nodes within the virtual space.

[5] This might take the form of attaching a physical sensor node to some sort of robotic device or even to an unmanned aerial vehicle that could move around a floor or in a 3-dimensional space. Such devices could also host an out-of-band wireless network that could communicate with the management backbone network used for the rest of the system.

The proposed distributed concept also allows arbitrary scalability. The next steps are the concrete implementation of the virtual mobility design by making use of existing implementations such as VirtualMesh as a basis for the VMI implementation, and WISEBED's virtual links.

Acknowledgements: The authors would like to acknowledge the support of the EC in funding this research under the Framework Programme 7. We would like specifically to thank COST Action IC 0906 for providing funding under the STSM scheme.

Bibliography

[AEGS10] N. Aschenbruck, R. Ernst, E. Gerhards-Padilla, M. Schwamborn. BonnMotion: a mobility scenario generation and analysis tool. In *SIMUTools '10: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. Pp. 1–10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, 2010.

[BCD$^+$10] T. Baumgartner, I. Chatzigiannakis, M. Danckwardt, C. Koninis, A. Kroller, G. Mylonas, D. Pfisterer, B. Porter. Virtualising Testbeds to Support Large-Scale Reconfigurable Experimental Facilities. In *in Proceedings of the 7th European Conference on Wireless Sensor Networks (EWSN 2010), eds. J. Sa Silva and B. Krishnamachari and F. Boavida (LNCS 5970)*. 2010.

[BCF$^+$10] T. Baumgartner, I. Chatzigiannakis, S. Fekete, C. Koninis, A. Kröller, A. Pyrgelis. Wiselib: A Generic Algorithm Library for Heterogeneous Sensor Networks. In Silva et al. (eds.), *Wireless Sensor Networks*. Lecture Notes in Computer Science 5970, pp. 162–177. Springer Berlin / Heidelberg, 2010.

[BJSS09] K. Borries, G. Judd, D. Stancil, P. Steenkiste. FPGA-Based Channel Simulator for a Wireless Network Emulator. In *IEEE 67th Vehicular Technology Conference (VTC2009-Spring)*. Barcelona, Catalunya, Spain, April 2009.

[CKM$^+$09] I. Chatzigiannakis, C. Koninis, G. Mylonas, S. Fischer, D. Pfisterer. WISEBED: an Open Large-Scale Wireless Sensor Network Testbed. In *Proceedings of the 1st International Conference on Sensor Networks Applications, Experimentation and Logistics*. Sept. 2009.

[DHK$^+$09] D. Dudek, C. Haas, A. Kuntz, M. Zitterbart, D. Krüger, P. Rothenpieler, D. Pfisterer, S. Fischer. A Wireless Sensor Network For Border Surveillance (Demo). In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. ACM, Berkeley, CA, Nov. 2009.

[ESHF04] M. Engel, M. Smith, S. Hanemann, B. Freisleben. Wireless ad-hoc network emulation using microkernel-based virtual Linux systems. In *5th EUROSIM Congress on Modeling and Simulation*. Pp. 198–203. Cite Descartes, Marne la Vallee, France, September 6-10 2004.

[JS03] G. Judd, P. Steenkiste. Repeatable and Realistic Wireless Experimentation through Physical Emulation. In *2nd Workshop on Hot Topics in Networks (Hot-Nets II)*. Boston, MA, USA, November 2003.

[KBHS07] T. Krop, M. Bredel, M. Hollick, R. Steinmetz. JiST/MobNet: combined simulation, emulation, and real-world testbed for ad hoc networks. In *WinTECH '07*. Pp. 27–34. ACM, New York, NY, USA, 2007.

[LWD+09] M. Lacage, M. Weigle, C. Dowell, G. Carneiro, G. Riley, T. Henderson, J. Pelkey. The Network Simulator ns-3. http://www.nsnam.org/, 2009.

[SBBD03] S. Sanghani, T. Brown, S. Bhandare, S. Doshi. EWANT: the emulated wireless ad hoc network testbed. Volume 3, pp. 1844 –1849 vol.3. mar. 2003.

[SGB11] T. Staub, R. Gantenbein, T. Braun. VirtualMesh: An Emulation Framework for Wireless Mesh and Ad-Hoc Networks in OMNeT++. *SIMULATION: Transactions of the Society for Modeling and Simulation International* 87(1-2):66–81, January 2011.

[Var01] A. Varga. The OMNeT++ Discrete Event Simulation System. In *European Simulation Multiconference (ESM'2001)*. Prague, Czech Republic, June 6-9 2001.

[WLG02] B. White, J. Lepreau, S. Guruprasad. Lowering the Barrier to Wireless and Mobile Experimentation. In *First Workshop on Hot Topics in Networks (HotNets-I)*. Princeton, New Jersey, USA, October 28-29 2002.

[WLS+02] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Fifth Symposium on Operating Systems Design and Implementation*. Pp. 255–270. Boston, MA, USA, December 2002.

[WSHW08] E. Weingärtner, F. Schmidt, T. Heer, K. Wehrle. Synchronized network emulation: matching prototypes with complex simulations. *SIGMETRICS Perform. Eval. Rev.* 36(2):58–63, 2008.

[WVBC09] B. Walker, I. D. Vo, M. Beecher, C. Clancy. A demonstration of the meshtest wireless testbed. *Testbeds and Research Infrastructures for the Development of Networks & Communities, International Conference on* 0:1, 2009.

[ZL02] Y. Zhang, W. Li. An integrated environment for testing mobile ad-hoc networks. In *3rd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '02)*. Pp. 104–111. ACM, New York, NY, USA, 2002.