A Hybrid Testbed for a Seamless Combination of Wireless Sensor Networks and OMNeT++ Simulations

Stefan Unterschütz
Institute of Telematics
Schwarzenbergstraße 95E
21073 Hamburg, Germany
stefan.unterschuetz@tu-harburg.de

Volker Turau
Institute of Telematics
Schwarzenbergstraße 95E
21073 Hamburg, Germany
turau@tu-harburg.de

Abstract—**Protocol development for large-scale wireless networks is challenging due to the inherently unreliable nature and massive distribution of such systems. Specially the transition from simulation to a real testbed deployment and later from testbed to large-scale deployments is frequently accompanied by many software and hardware errors. In this paper a hybrid testbed approach is introduced which seamlessly combines real 802.15.4 deployments with real-time simulations based on OMNeT++. The integration of a simulation tool in a testbed bears good prospects and reduces costs since extensive testbed evaluation can be avoided. The transition between execution environments is completely transparent to the involved nodes, which only perceive a homogeneous wireless mesh network. The effort for the programmer is kept low since no code adaptation is necessary for running software in mixed environments.**

I. Introduction

In traditional methods for the development of wireless sensor networks (WSN) simulations and testbed deployments are used to evaluate the system. Here, the transition from simulation to testbed and testbed to large-scale deployments is challenging since unexpected problems can render the whole system unusable. In such situations testing and debugging is difficult due to the distributed nature of WSNs.

A common problem of most wireless networks is the lack of reusable software components, i.e. communication protocols. For example base station, gateways and wireless nodes rarely share the same implementation of protocols or even the same programming language. Hence, additional effort for implementation and testing becomes necessary for each execution environment. In this context, an execution environment specifies the platform and environmental condition in which software runs.

This paper introduces the design of a hybrid testbed for a seamless interconnection of different execution environments. For this purpose we use the CometOS framework [1] which provides an interface for running code platform independently on wireless, resource-restricted nodes, desktop systems (e.g. Linux, Windows), or even directly in the OMNeT++ network simulator. All these execution environments can seamlessly be interconnected to a virtual, homogeneous network. This provides the possibility to extend simulations by real hardware in order to allow a step by step transition from simulation to a large-scale deployment. This enhances testing and debugging of WSNs.

The hybrid testbed approach is already applied in two projects, the HelioMesh and iEZMesh project, and thus we will exemplary refer to these in the following. In the HelioMesh project [2] wireless communication based on IEEE's 802.15.4 standard is used to control heliostats of a solar power plant. For this purpose a base station either communicates to heliostats directly or by routing over intermediate nodes. In iEZMesh[1] we pursue the goal to retrieve consumption data of smart meters using wireless communication. The data is forwarded using the dense distribution of smart meters in urban regions to a gateway in which it is sent to the corresponding supplier of electric energy.

This paper is structured as follows: Section II gives an overview of current operating systems and simulation environments for WSNs which have the capacity to provide the required functionality. Furthermore, existing approaches of hybrid testbeds for IP-based networks are described. In Sect. III the use cases and the execution of code in different environments are introduced and discussed. Finally, the paper ends with a short conclusion.

II. Related Work

In this chapter well-known operating systems and simulation tools are described which come into consideration for building up a hybrid testbed. Afterwards, we review existing hybrid testbeds for IP-based networks.

OMNeT++ is an object-oriented, discrete event simulation framework [3] written in C++. It uses message-passing for the communication between software modules. This approach provides a very loose coupling and greatly promotes the development of reusable building blocks. While OMNeT++ itself does not provide specific components to simulate a certain problem domain, a large number of model frameworks have emerged addressing this issue. For

[1]iEZMesh: http://www.ti5.tu-harburg.de/research/selfstab/iezmesh/

example, the MiXiM framework [4] aims at the simulation of wireless sensor networks and provides extensive support for modeling the wireless communication channel, mobility of nodes, MAC layers and energy consumption. Another extension for OMNeT++ is the CometOS framework [1] which provides an abstraction layer that allows code execution in OMNeT++ as well as on real sensor node hardware.

TOSSIM is a code level simulator for TinyOS applications written in the nesC programming language [5], [6]. It provides physical layer modeling at Bit granularity, inspection of base-type variables and the visualization tool TinyViz. Experiments are configured using the Python scripting language. Some drawbacks of TOSSIM are the limitation to a single code image and limited debugging capabilities—as there is no debugger for nesC, one has to debug the generated C code and mentally translate identifiers. Another popular OS for wireless sensor networks, Contiki, comes with the COOJA/MSPSim simulator [7]. Its highlight is the so-called cross-level simulation. Thereby, nodes within the same network can be simulated at different levels: at network level (a high-level Java implementation), at operating system level (an actual Contiki-application) and at machine code instruction level (a per-instruction hardware emulation).

Tinyos and Contiki have a basic architecture which might enable a testbed/simulation execution requiring some implementation effort. However, the message passing architecture of OMNeT++ and CometOS and the loose coupling of modules are highly suited for the intended project. Next to connecting heterogeneous networks, even an distributed execution of a node's protocol stack becomes possible.

TWINE is a framework that combines simulation, emulation and physical networks in an integrated testbed used for the evaluation of wireless 802.11 IP-based networks [8]. A similar project is Hybrid MCG-Mesh [9]. This hybrid testbed consists of a real network and a virtualization environment. Both approaches are based on an existing IP-based infrastructure and are not suited for resource-restricted sensor nodes and 802.15.4 networks.

III. System

In this section, our concept for a seamless combination of execution environments for WSNs is explained. Firstly, use-cases of a hybrid testbed and their benefits are introduced. Then relevant parts of the CometOS framework are described which are crucial for our concept. Afterwards, the realization of the hybrid testbed is depicted and discussed.

A. Use-Cases

The protocol stack of a single sensor node can be distributively executed. Figure 1 shows a protocol stack containing an application layer, a routing protocol, and a MAC layer. The latter two are simulated in OMNeT++, whereas the application runs on a real sensor node. This scenario permits rapid prototyping of single modules in

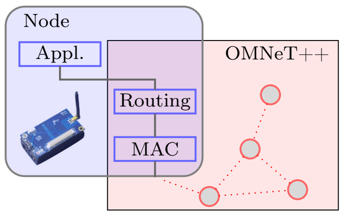

Figure 1: Running a single protocol stack in different execution environments.

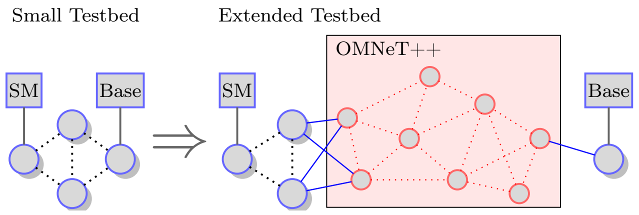

Figure 2: Extending a small testbed to a large-scale network.

real hardware. In the iEZMesh this approach allows reading consumption data from a smart meter with real hardware. Routing and processing of these data is completely done in the simulation. This experiment allows us to give dedicated statement how routing protocols work with real traffic.

The step from small testbeds to large-scale deployments is challenging. Higher contention, an increased neighborhood size, unreliable links can render the whole system unusable. To cushion this transition it is possible to extend a small testbed to a large-scale network as shown in Fig. 2. Initially, a small testbed containing a base station (Base), a smart meter (SM) and four nodes are connected. This network is split in two sets and extended by a large-scale intermediate network running in OMNeT++. This scenario is highly appropriated for analyzing scalability. It is important to mention that a real sensor node can be connected to several adjacent simulated nodes.

The concepts of the hybrid testbed can be adopted to hide complexities of heterogeneous infrastructures. In the HelioMesh project the use of multiple gateways become necessary to cope with the funneling effect. These gateways are connected via TCP/IP with a central base station. The existence of heterogeneous devices often makes an adaptation of the software necessary as well as special glue code. Our systems inherently masks such infrastructural measures as shown in Fig. 3. Furthermore, the base station and the wireless nodes can share the same implementation of most modules, e.g. network protocols, without any porting effort.

B. CometOS

The base of the hybrid testbed is CometOS which uses C++ as primary programming language. This framework has the following three properties which are fundamental for our approach: All protocol implementations are encap-

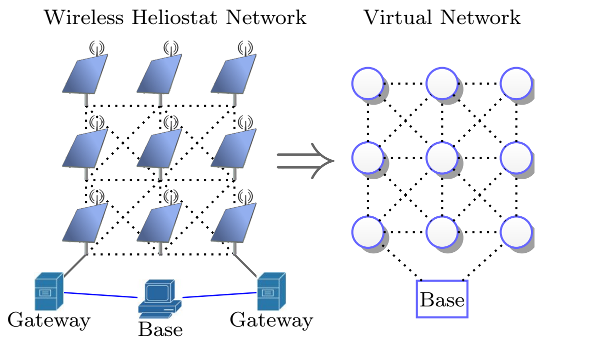

Figure 3: Masking an underlying infrastructure of wireless sensor networks.

sulated in separate modules (modularity). Most modules can be compiled for sensor nodes, desktop systems, and OMNeT++ (platform-independence). Inter-module communication is realized via message-passing which leads to a loose coupling of modules. The properties modularity, platform-independence, and message-passing provide a good starting point for the setup of a hybrid testbed in which different environments must be connected.

To further decrease coupling of modules we defined general purpose messages. In the protocol hierarchy, an upper layer, which is willing to transmit data, sends a data request to the lower layer and may receive a confirmation as reply. A lower layer uses the message type data indication to pass a received message to the upper layer, which may reply with a data response. These messages are similarly defined for service access points of the OSI reference model. In our implementation data request and indication messages contain address fields to specify a destination and source as well as arbitrary payload. The benefit of general purpose messages is the flexibility in module usage. For example, an application can be placed directly above a MAC layer, network layer, or transport layer without the need of modifications of the interfaces. For the sake of completeness, it should be mentioned that CometOS provides the attachment of meta data to each packet in order to support cross-layering.

In CometOS all objects that are appended to the payload of a message are serialized. The code for serialization and deserialization is shared among all supported platforms. As a result inter-module communication is not limited to the execution on a single system.

C. *Realization*

The lowest layer of a node's communication stack is a network interface which provides access to adjacent nodes. The type of the network interface depends on the role and kind of the node. To set up the scenarios described in III-A we use the following network interfaces:

- TcpComm connects IP networks. It provides necessary functionalities to run a server and to build up several client connections to other instances of TcpComm. This module runs on desktop systems and OMNeT++.
- SerialComm enables a serial communication between two devices. For the message transmission acknowledgments and check sums are used to ensure the correct reception of data. SerialComm runs on all platforms.
- The MacAbstractionLayer is a common interface to access the wireless channel. It provides a convenient base for a variety of MAC implementation such as CSMA, TDMA, or duty-cycled protocols. The MacAbstractionLayer is available for sensor nodes. For example, in the HelioMesh project we used a 802.15.4 conform implementation. In OMNeT++ the MiXiM framework is used to realize the MacAbstractionLayer.
- The NetworkInterfaceSwitch is a proxy layer that is connected to a Common Network Interface (CNI) and a Network-to-Network Interface (NNI). The former is used to send messages among nodes of the same network. NNI is intended to send messages from one execution environment to another. The destination address of the packet is consulted to determine which network interface to use. For this purpose the user must specify the address space for CNI and NNI, as well as the broadcast address which is used to transmit messages via both network interfaces.

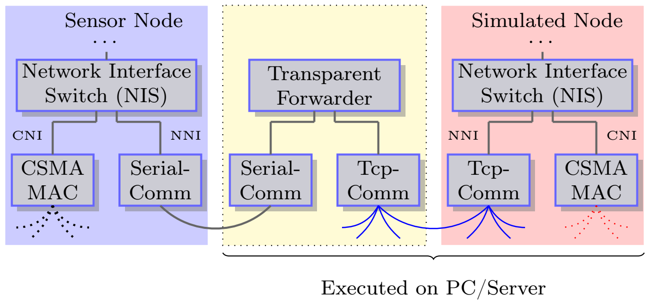

Figure 4: Connecting nodes in different networks.

A possible setup of a hybrid testbed is shown in Fig. 4. For the sake of clarity, all upper modules are not depicted except of the network interface. The sensor node uses a CSMA MAC layer and the SerialComm module. In our testbeds all messages to a destination with an identifier less than 0x1000 are passed to NNI. Hence, the maximum number of simulated nodes is limited to 4096 which is sufficient since the number of nodes that can be simulated in real-time is limited to a few hundred due to the high computation effort. As shown in the figure the sensor node is not directly connected to the simulated node, furthermore all messages are send to the TransparentForwarder which directly forwards all data to the simulated node using TcpComm. In contrast to SerialComm, TcpComm provides multiple connections to other devices. The existence of the TransparentForwarder is not perceivable by either the sensor node nor the simulated node. Note that the NNI of the simulated node uses addresses between 0x1000-0xfffe in order to send data back into the real sensor network. For broadcasts the address 0xffff is taken which leads to the use of both network interfaces.

A distributed execution of a single protocol stack as shown in Fig. 1 is also realized by a combination of the SerialComm and TcpComm module. This is easily possible since all protocols are using the same communication messages, i.e. data request and data indication.

D. Discussion

We refrained from using a well-known operating system such as TinyOS for our hybrid testbed since we believe that CometOS as explained in III-B is more suited. For example, TinyOS uses Java applications on desktop systems to communicate with sensor nodes that are programmed in NesC. To provide a hybrid testbed, a high configuration and implementation effort is needed.

The maximum number of nodes which can be simulated in real-time by one instance of OMNeT++ is limited depending on the amount of traffic and topology, i.e. the node density. In case of an overloaded simulation the latency increases since simulation time is not any longer coherent to the real-time. In our experiments 100 nodes are the limit for running traffic intensive protocols such as pure flooding. Of course, it is possible to interconnect multiple OMNeT++ networks to increase the node count.

Another important issue that must be taken into consideration is the increased latency and specific characteristics of each network interface. Packet loss in wireless networks often occur with a higher probability then using serial communication. Additionally, the transmission time of a message depends on the data rate of the network interface. Also the occurrence of collisions differs. However, protocols used in a hybrid testbed should not rely on strict time constraints or certain channel characteristics.

In experiments we determined the latency introduced by the hybrid testbed. For this purpose we measured the round-trip time for transmitting and consecutively receiving 60 Bytes of data over different network interfaces. The experiment was repeated 1000 times. The round-trip time between two sensor nodes is 9.568 ms using the CSMA MAC. Simulated nodes only need 8.750 ms. SerialComm with a baud rate of 57600 needs 24.723 ms and TcpComm requires 14.459 ms. Note that in SerialComm each packet transmission is acknowledged which lowers the achievable throughput. TcpComm is executed in the synchronized context of CometOS and OMNeT++. Sockets are checked for any new data each 5 ms which leads to the high round-trip time. Threads are not used due to the fact that OMNeT++ is not thread-safe. Obviously, the transition from a real to a simulated network has a high latency. We believe that optimizing TcpComm can significantly decrease these latencies.

IV. Conclusion

This paper introduced a hybrid testbed for extending a real testbed deployment by an OMNeT++ simulation. The system further provides the ability of a distributed execution of a single protocol stack as well as a convenient base to set up more complex infrastructure for wireless sensor networks. The used CometOS framework allows code reuse in different sensor network platforms, desktop systems, and OMNeT++.

The hybrid testbed is successfully applied in two industrial projects. Here, the simulation support provided us excellent testing, debugging, evaluation options for rapidly prototyping software for large-scale deployments. The hybrid testbed support will be part of the next CometOS release.

References

[1] S. Unterschütz, A. Weigel, and V. Turau, "Cross-Platform Protocol Development Based on OMNeT++," in *OMNeT++ '12: Proc. 5th International Workshop on OMNeT++*, Mar. 2012.

[2] S. Kubisch, M. Randt, R. Buck, A. Pfahl, and S. Unterschütz, "Wireless Heliostat and Control System for Large Self-Powered Heliostat Fields," in *SolarPACES '11: Proc. 17th International Symposium on Concentrated Solar Power and Chemical Energy Technologies*, Sep. 2011.

[3] A. Varga, "The OMNeT++ Discrete Event Simulation System," in *ESM '2001: Proc. 15th European Simulation Multiconference*, Prague, Czech Republic, Jun. 2001.

[4] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. K. Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin, "Simulating Wireless and Mobile Networks in OMNeT++ the MiXiM Vision," in *Simutools '08: Proc. first International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, Mar. 2008.

[5] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A holistic Approach to Networked Embedded Systems," *SIGPLAN Not.*, vol. 38, May 2003.

[6] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An Operating System for Sensor Networks," in *EUSAI '04: Proc. second European Symposium on Ambient Intelligence*, Nov. 2004.

[7] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," in *LCN '06: Proc. 31st IEEE Conference on Local Computer Networks*, Nov. 2006.

[8] J. Zhou, Z. Ji, and R. Bagrodia, "TWINE: A Hybrid Emulation Testbed for Wireless Networks and Applications," in *INFOCOM '06: Proc. 25th Conference on Computer Communications*, Apr. 2006, pp. 23–29.

[9] A. Zimmermann, M. Günes, M. Wenig, J. Ritzerfeld, and U. Meis, "Architecture of the hybrid MCG-Mesh Testbed," in *WiNTECH '06: Proc. 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, Sep. 2006, pp. 88–89.