Poster Abstract: A 6LoWPAN Model for OMNeT++[*]

Michael Kirsche[†] and Jonas Hartwig
Computer Networks and Communication Systems Group
Brandenburg University of Technology (BTU) Cottbus, Germany
[michael.kirsche, jonas.hartwig]@tu-cottbus.de

ABSTRACT

This paper introduces a 6LoWPAN simulation model for OMNeT++. Providing a 6LoWPAN model is an important step to advance OMNeT++-based Internet of Things simulations. We integrated Contiki's 6LoWPAN implementation into OMNeT++ in order to avoid problems of non-standard compliant, non-interoperable, or highly abstracted and thus unreliable simulation models. The paper covers the model's structure as well as its integration and the generic interaction between OMNeT++ / INET and Contiki.

Categories and Subject Descriptors

I.6.5 [**Computing Methodologies**]: Simulation and Modeling - Model Development; C.2.2 [**Computer Systems Organization**]: Computer Communication Networks - Network Protocols

Keywords

6LoWPAN, Simulation Model, OMNeT++, Contiki

1. INTRODUCTION

The coupling of Wireless Sensor Networks (WSNs) and embedded systems with the Internet is an emerging trend, forming the Internet of Things (IoT) movement. Integrating stand-alone devices, sensors, and actuators *(things)* enables an interconnection between the pure virtual Internet and the physical world we live in. Technical advances like small-scale yet fully-compliant IP stacks (e.g., uIP [2]) and adaptation layers like 6LoWPAN [6] enable this integration of IPv6-powered sensor networks. While practical IoT deployments continue to emerge regularly, simulation support for certain IoT-related protocols is still an open topic.

[*]The source code and additional documentation for the presented 6LoWPAN simulation model can be found at: `http://github.com/michaelkirsche/6lowpan4omnet-diy`
[†]Corresponding author.

IoT scenarios typically combine communication protocols and standards like IEEE 802.15.4, IEEE 802.3, IPv6, TCP, UDP, or RPL[1] below the application layer. OMNeT++ supports most of these protocols through extension frameworks such as INET[2] and MiXiM[3], although models for important new protocols like 6LoWPAN or RPL are missing. This work focuses on IPv6 over Low-power Wireless Personal Area Networks (6LoWPANs), which specify how to deploy IPv6 in resource constrained networks with IEEE 802.15.4 low data rate transceivers. Available implementations [3] enable a practical deployment of IPv6 in sensor networks and thus a combination of them and the IP-driven Internet. An accurate 6LoWPAN simulation model is still necessary for various reasons. One could argue that a static reduction of the IP header size is an adequate and sufficient way to “simulate” 6LoWPAN. However, this abstraction is inaccurate, because the level of header compression can vary inside a network, depending on the spreading of shared context information, actually deployed upper layer protocols, and device abilities. Instead, we need a feature-complete simulation model to further analyze the interconnection of WSNs with the Internet and pave the way for the creation of missing simulation models (e.g., RPL). This work introduces a 6LoWPAN model for OMNeT++, based on the integration of a tested implementation from the Contiki [1] operating system for memory-constrained and networked devices into OMNeT++ for use with INET/MiXiM.

The following section introduces our chosen integration approach, the 6LoWPAN simulation model, its structure, and the interaction between Contiki and OMNeT++. Instructions regarding the integration and testing of the model are given in Section 3, followed by a discussion of possible validations. Final remarks conclude this work in Section 4.

2. THE 6LOWPAN SIMULATION MODEL

We chose to integrate an existing 6LoWPAN implementation into OMNeT++ rather than modeling the protocol from scratch. In sum, our approach is to embed integral parts of Contiki into OMNeT++ by compiling Contiki as a library with its interfaces mapped to OMNeT++. This enables the use of Contiki implementations like 6LoWPAN directly in OMNeT++, thereby extending OMNeT++'s IoT simulation capabilities. After explaining our motives, we provide more details of the integration and structure of the model.

[1]Routing Protocol for Low power and Lossy Networks
[2]`http://inet.omnetpp.org/`
[3]`http://mixim.sourceforge.net/`

Motivation and Reasons

We start by explaining why we chose to integrate parts of Contiki into OMNeT++ before going deeper into the actual integration. Today's options for simulating Internet of Things scenarios are manifold but incomplete. Sensor node operating systems like TinyOS and Contiki provide their own simulators (e.g., Cooja [5]). Although Cooja is a handy tool with various strong points in the exact simulation of sensor nodes at runtime (due to the execution of the actual deployed source code), it still is a simulation environment with a scope limited to wireless communication between sensor nodes. Complex IoT scenarios include diverse protocols as mentioned before, but participating systems and networks also vary. Generic simulators like OMNeT++ have their advantages when it comes to the simulation of diverse networks and protocols due to their extensive simulation and extension libraries. Extension frameworks like MiXiM and INET provide almost all necessary protocols for todays Internet-based IoT scenarios. Cooja simulations only model a sensor network, which is often the last cornerstone of a complex IoT scenario; vital, of course, but nevertheless not sufficient in many cases. If, for example, different sensor networks with separated border gateways are combined in a scenario with Internet-based back end systems, then OMNeT++ is, simply said, our only weapon of choice. That is, if we can provide the means to model and simulate the cornerstone sensor network and its integration into the Internet accurately. With the 6LoWPAN model, we hope to close this existing gap and enhance OMNeT++'s IoT simulation capabilities.

A second motive is to use a tested implementation rather than model 6LoWPAN from scratch. Avoiding abstraction inaccuracies and modeling errors are two of the obvious reasons. We also aim for a long-term contribution, as this integration of 6LoWPAN can be seen as an initial step for a generic integration of Contiki instances into OMNeT++. Our motivation is based on the fact that, although OMNeT++ extension frameworks like Castalia provide accurate models for typical WSN applications, their inability to use common Internet-based protocols or IP stacks is a considerable drawback for IoT scenarios. By integrating (parts from) Contiki, we want to enhance the accuracy of frameworks like INET, which were primarily developed for non-WSN scenarios. As this is still pending work, we are unsure if a complete integration is either possible or profitable.

Contiki as a Fundament

In-depth comparisons of 6LoWPAN implementations were made in [3, 8]. We chose Contiki's implementation, as it is written in plain C, which eased a combination with OMNeT++ when compared to TinyOS and its nesC language. Contiki's 6LoWPAN feature set is implemented in accordance with RFC 4944 [4]. Supported features include 64-bit addressing, a full fragmentation support, stateless and stateful header compression, and neighbor discovery (currently without a distribution of context information for context-based header compression). Recent enhancements [7] for the neighbor discovery in 6LoWPANs are not yet implemented.

We analyzed Contiki's 6LoWPAN code to identify integration strategies. A direct integration (complete port) of the 6LoWPAN source code from Contiki (`C`) to OMNeT++ (`C++`) was turned down because massive code changes would have been necessary and future changes in Contiki / 6LoWPAN would have to be ported again, requiring an extensive maintenance. Another integration approach is a statically compiled and linked bridge between OMNeT++ and Contiki. Drawbacks are a complex memory management for Contiki instances and the issue of synchronizing communication over the bridge. Instead, we chose to extend Contiki with a new platform: `omnetpp`. Contiki interface calls are redirected and mapped to OMNeT++ in this platform. Memory management of 6LoWPAN instances running in OMNeT++ is done comparable to Cooja. We save the runtime configurations of Contiki instances in OMNeT++ whereas the exact instance gets identified by its gate ID to support simulations of multiple Contiki instances. Building an empty Contiki program with uIP support for the `omnetpp` platform and linking the created library to OMNeT++ enables the use of 6LoWPAN in OMNeT++ without major changes in the 6LoWPAN code itself. Changes of Contiki's interfaces must be retraced in the code of the 6LoWPAN model while other source code changes can just be "ported" by recompiling Contiki.

The 6LoWPAN model itself consists of the following parts:

- the Contiki platform source code (`omnetpp`) and the associated `Makefile`,
- a Contiki configuration file and the appropriate configuration options for the 6LoWPAN part,
- the 6LoWPAN "wrapper" (`_6lowpan`) for OMNeT++, integrated in the INET framework.

An integration into INET can be done, for example, directly inside the IPv6 standard host. Figure 1 depicts the network layer of the IPv6 host and the integrated 6LoWPAN wrapper (`cSimpleModule sixlowpan`).

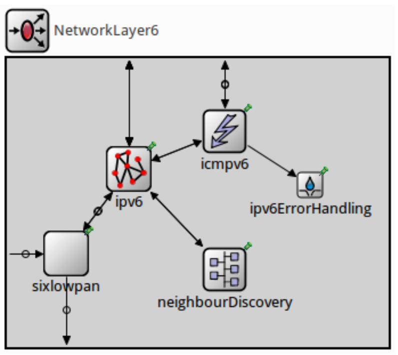

Figure 1: Network layer of an INET IPv6 node with the integrated 6LoWPAN wrapper.

Modus Operandi

The module's modus operandi is as follows. First, the module and the instance memory management are initialized, while interfaces (OMNeT++ gate connected to MAC layer) are checked for IEEE 802.15.4 capabilities provided through OMNeT++ extension frameworks. Whenever IPv6 packets arrive through the connected gate from the upper layer, they are converted into Contiki's format and written into a packet buffer. Transport layer protocols (TCP, UDP, ICMP) are also converted into Contiki's data format during this process. IPv6 extension headers and other transport protocols are not yet supported by the wrapper. Transport layer packets are then encapsulated into 6LoWPAN packets.

Contiki’s 6LoWPAN input function is called, whereas Contiki’s 6LoWPAN code handles and (optionally) fragments the packet and compresses the header information when applicable. Required link-local addresses of next hops are determined by the `neighbourDiscovery` module (see Fig. 1). Calling Contiki’s input function and using Contiki resources from OMNeT++ is possible because (i) according interfaces between Contiki and OMNeT++ are defined inside a `C` file (`contract.c`), which is compiled into the Contiki library, and (ii) through the use of the `extern "C"` directive and the subsequent inclusion of Contiki header files in the OMNeT++ module. The contract file contains the function prototypes (`init`, `send`, `input`, `on`, and `off` amongst others) that are substituted and set to static OMNeT++ functions during the wrappers’ initialization in OMNeT++. Generated and processed packets are saved in a static queue and then sent to the lower (MAC) layer through OMNeT++ messages.

When a node receives a packet from the IPv6 module, it also checks if a context for the stateful header compression must be added. If the context-based compression is enabled (via Contiki `#define`), then each node adds the necessary context separately. Nodes add their prefix once they have obtained their global IP addresses.

When a message is received on the lower layer, it is processed by Contiki’s network stack (i.e., the 6LoWPAN part) and the resulting packet is converted from Contiki’s IPv6 format into an INET packet, including the transport layer headers. Possible fragments have to be treated separately during this process. The instance memory management is used here to save the state / configuration of each 6LoWPAN instance for each reassembly process, comparable to the treatment in Cooja. The saving of the state also allows OMNeT++ to monitor the state of reassembly and the values of important status variables. The processed packet is then forwarded to the upper layer (i.e., IPv6).

3. INTEGRATION AND TESTING

The practical use of the 6LoWPAN simulation model is the subject of this section. Subsections deal with the integration and use of the model, the combination with OMNeT++ extension frameworks, important parameters, test approaches, and a concluding discussion of evaluation ideas.

Integrating the Model

To integrate 6LoWPAN, we compile Contiki with *uIPv6* support for the newly created `omnetpp` platform, to enable the described interface mapping. In addition, Contiki’s and OMNeT++’s clock need to be mapped. This is done in a way comparable to the definition of interfaces in the `contract.c` file by making Contiki’s clock count available in OMNeT++. We synchronize both clocks with the preprocessor directive `CLOCK_CONF_SECOND` (adjusted to μs). When using the model in OMNeT++, the simulation time scale is adjusted to microseconds (set `simtime-scale` to `-6` in `omnetpp.ini`).

Before the actual OMNeT++ simulation or the compilation of the extension framework, the created Contiki library and Contiki’s header files must be integrated into the build process. This can be done with the help of an additional makefile, referenced in INET’s `makefrag` file for example, where paths for the headers to include are specified along with the path and the name of the Contiki library. External resources are then made available with the `extern "C"` directive inserted in the header file of the 6LoWPAN `cSimpleModule`. All necessary Contiki header files must be included in the `extern "C"` block, together with definitions of all resources that are not specified in any included header. Last but not least, small changes in the 6LoWPAN implementation in Contiki itself are unavoidable. These changes only affect the visibility of variables: certain local variables (required for the fragmentation process) must be changed into global variables by removing the `static` keyword, so that they can be saved and accessed from within OMNeT++. Afterwards, the model can be compiled for use with OMNeT++ / INET.

Using the Model

The first release of the 6LoWPAN model can be used with *INET*, *INETMANET* and *MiXiM* (plus *mixnet* extension). Figure 2 depicts example configurations of 6LoWPAN capable hosts for *INETMANET* and *MiXiM/mixnet*.

INETMANET[4] is an INET4 branch with additional modules for mobile ad hoc networks and a port of an older IEEE 802.15.4 model. Combining the IEEE 802.15.4 model from INETMANET and INET IPv6 capabilities (support for packet transfer, addressing, header and frame formats, stateless auto-configuration, and neighbor discovery) is possible without problems. MiXiM, on the other hand, does not support IPv6 because it was specifically developed for lower layer simulations of wireless systems. However, the *mixnet* project extends MiXiM with the ability to combine its network interface cards (i.e., MiXiM’s lower layer models) with the higher layer models from INET4 (e.g., IPv6, TCP, UDP). The necessary INET4 branch without wireless modules is available online[5]. Mixnet provides a bridging functionality that enables the use of IPv6 from INET4 over an IEEE 802.15.4 network from MiXiM. Figure 2 depicts this bridge. INET4, as common ground for both frameworks in terms of IP protocols, is therefore an ideal candidate to integrate our 6LoWPAN model into. 6LoWPAN simulations are thus available for a wide range of use cases, supporting many of the available OMNeT++ frameworks and libraries.

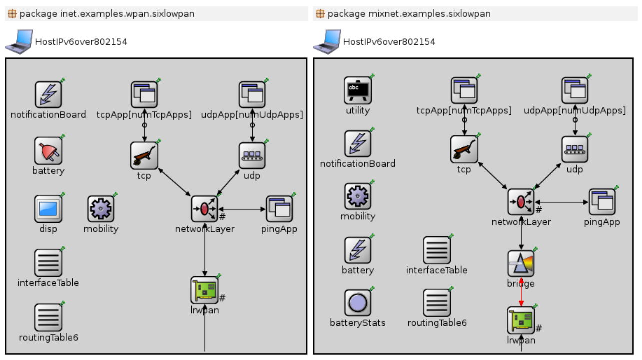

Figure 2: 6LoWPAN-capable IPv6 hosts in INETMANET and MiXiM (with mixnet extension).

Important Configuration Parameters

Table 1 summarizes important configuration options for the 6LoWPAN implementation. These options are available in the `omnetpp` platform configuration file `contiki-conf.h`.

[4]`http://github.com/inetmanet/inetmanet`
[5]`http://github.com/mixim/inet4mixnet`

Parameter	Default	Description
`UIP_CONF_IPV6`	1	Use IPv6 or not.
`UIP_CONF_UDP`	1	Toggles whether UDP support is compiled in or not.
`UIP_CONF_TCP`	1	Toggles TCP support.
`UIP_CONF_ICMP6`	1	Toggles ICMPv6 support.
`SICSLOWPAN_CONF_COMPRESSION`	2 (HC06)	Switching between available compression types (i.e., IPv6, HC01, HC06).
`SICSLOWPAN_CONF_FRAG`	1	Toggle support for 6LoWPAN fragmentation.
`SICSLOWPAN_REASS_MAXAGE`	(e.g.) 20	Adjusting the timeout for the 6LoWPAN packet reassembly.
`CLOCK_CONF_SECOND`	1000000	Synchronization between Contiki's and OMNeT++ clock's (in μs).
`NETSTACK_CONF_MAC`	`omnet_mac_driver`	MAC driver, Redirect input/output calls from OMNeT++ to Contiki.
`NETSTACK_CONF_NETWORK`	`sicslowpan_driver`	Network driver, enable the 6LoWPAN adaptation layer in Contiki.

Table 1: Extract of parameters of the 6LoWPAN model (adjustable in Contiki's configuration file).

Testing the Model

After compiling the model and creating an example set-up, we have to verify the correct functionality of the model. It is important to check that parameters like the Maximum Transmission Unit (MTU) are consistent between Contiki and OMNeT++ / INET, or else data will be cut and unpredictable behavior will occur. A user should enable the logging features at first to check if address compression and especially fragmentation are working correctly. Compression results can be checked easily for ICMPv6 ping messages and their headers as well as for TCP and UDP headers separately. Example set-ups for INETMANET and MiXiM will be published on the website of the 6LoWPAN model.

Validation

Every simulation model must be validated to strengthen its credibility. The 6LoWPAN model can be validated in several ways. First and foremost, a correct functionality must be proven. This is done for compression and fragmentation operations in accordance with the expected results based on the standard specification [4]. Validation could also be done against hardware and real life deployments of Contiki as well as comparisons against Contiki's own simulator, Cooja. Furthermore, the generic cooperation between Contiki and OMNeT++ could be validated in terms of scalability, run time, and memory consumption.

These tasks are pending work. We are in the process of comparing the model against Contiki deployments and Cooja. Although both steps are crucial for the model's credibility, they prove to be complicated because the underlying hardware capabilities can alter expected results. We observed, for example, that sensor nodes (e.g., Atmel Raven vs. Redbee Econotag) produce different results for lower layer fragmentation. Validating against Cooja also proves to be complicated without the right hardware, as Cooja requires representations of the underlying hardware. We are currently switching to the Zolertia Z1 platform, which is now supported under Cooja, to validate a 6LoWPAN deployment against an OMNeT++-driven 6LoWPAN simulation.

Further research directions for us are a generic integration of Contiki instances into OMNeT++ and a validation of this integration, which, when validated, could provide OMNeT++ with a useful add-on for wireless sensor network simulation.

4. FINAL REMARKS

OMNeT++ already provides many integral parts for a simulation of scenarios in the Internet of Things context. With this work, we introduced a 6LoWPAN simulation model and our motivation behind this integration of a 6LoWPAN implementation from Contiki into OMNeT++, respectively INET. As the model is based on a real life implementation, it can hopefully achieve credibility in the long term.

Future work lies in the area of practical testing and validation. Furthermore, the enhancements [7] for 6LoWPAN neighbor discovery need to be integrated in Contiki and our 6LoWPAN model. As we mentioned, the 6LoWPAN integration can be seen as a first step for a complete integration of Contiki instances into OMNeT++. Further work along this way will show if integrating more parts than the presented 6LoWPAN implementation will enhance OMNeT++'s status in the IoT simulation community.

5. REFERENCES

[1] A. Dunkels, B. Groenvall, and T. Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proc. of the IEEE Workshop on Embedded Networked Sensors (Emnets)*, Nov. 2004.

[2] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels. Making Sensor Networks IPv6 Ready. In *Proc. of the 6th ACM Conference on Networked Embedded Sensor Systems (SenSys 2008), poster session*, Nov. 2008.

[3] Y. Mazzer and B. Tourancheau. Comparisons of 6LoWPAN Implementations on Wireless Sensor Networks. In *Proc. of the 3rd Conference on Sensor Technologies and Applications (SENSORCOMM)*, pages 689–692. IEEE Computer Society, 2009.

[4] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, IETF, September 2007.

[5] F. Oesterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-Level Sensor Network Simulation with COOJA. In *Proc. of the 1st IEEE Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, Nov. 2006.

[6] Z. Shelby and C. Bormann. *6LoWPAN – The Wireless Embedded Internet*. John Wiley and Sons Ltd, 2009.

[7] Z. Shelby, S. Chakrabarti, E. Nordmark, and C. Bormann. Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 6775, IETF, Nov. 2012.

[8] R. Silva, J. S. Silva, and F. Boavida. Evaluating 6lowPAN Implementations in Wireless Sensor Networks. In *Proc. of the 9th Conferencia sobre Redes de Computadores (CRC '09)*, 2009.