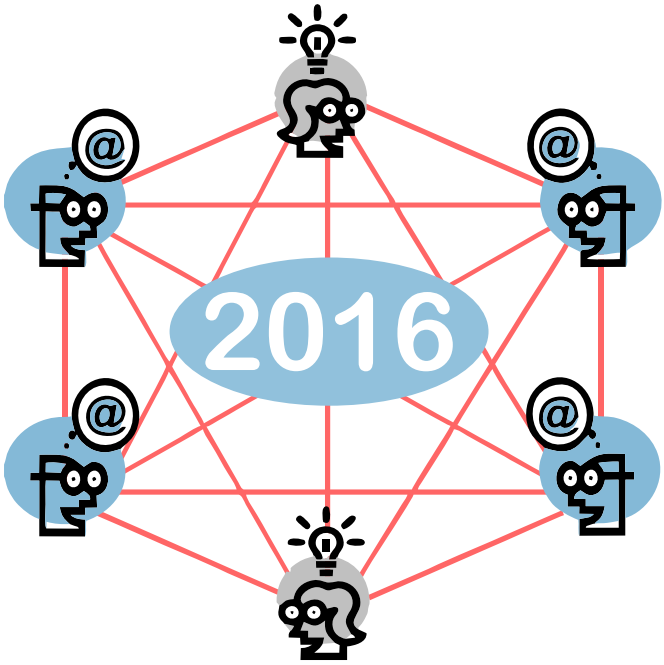
OMNeT++ Community Summit 2016

Successor of the International Workshop on OMNeT++

Brno University of Technology

– Czech Republic –

September 15-16, 2016



Proceedings of the 3rd OMNeT++ Community Summit

Community Summit Organizers

Founding Chair

Andras Varga (OpenSim kft.)

Community Summit Organizers

Anna Förster (University of Bremen, Germany)

Vladimír Veselý (Brno University of Technology, Czech Republic)

Technical Program Organizers

Antonio Virdis (University of Pisa, Italy)

Michael Kirsche (BTU Cottbus–Senftenberg, Germany)

Publicity Organizer

Kyeong Soo (Joseph) Kim (Xi'an Jiaotong–Liverpool University, China)

Laura Marie Feeney (Uppsala University, Sweden)

Technical Program Committee

Michael Frey (HU Berlin, Germany)

Yutaka Matsubara (Nagoya University, Japan)

Cyriel Minkenberg (Rockley Photonics Inc., USA)

Georg Panholzer (Salzburg Research, Austria)

Alfonso Ariza Quintana (University of Malaga, Spain)

Gabrielle Romaniello (Schneider Electric, France)

Christoph Sommer (University of Paderborn, Germany)

Dora Spenza (Sapienza University of Rome, Italy)

Mirko Stoffers (RWTH Aachen University, Germany)

Carlo Vallati (University of Pisa, Italy)

Welcome Note from the OMNeT++ Community Summit Organizers

The third edition of the OMNeT++ Community Summit took place in September 2016 at the Brno University of Technology in the Czech Republic.

The aim of the OMNeT++ Community Summit is to provide a forum for discussions on recent developments and novel ideas in the broad area of network simulation and modeling, with a focus on the OMNeT++ simulation environment. After six successful editions of the ACM/ICST International Workshop on OMNeT++ that started back in 2008, we decided to switch from the standard scientific workshop format to a new and open (access) format. This new format allows us to better encompass more visionary ideas and foster interaction between participants. The first Summit in 2014 in Hamburg, Germany, and the second Summit in Zurich, Switzerland, were both very successful with a vibrant program of keynotes, tutorials, discussion panels, demonstrations, and presentations about scientific research. The 2016 event continued to expand the idea of an open summit by addressing the interaction through tutorials and panels while keeping the extended 2-day time frame of the 2015 summit. We also continued to provide a peer-reviewing process to increase the scientific background and to give submitters feedback on their work.

OMNeT++ is a public-source, component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel. It is designed to simulate discrete event systems, but the primary application area is the simulation of communication networks. This is made possible by an ecosystem of simulation module libraries focusing on Internet protocols, vehicular and wireless communication networks, routing protocols and overlay networks, to name a few. They are designed, built, and curated by expert communities of researchers from their respective fields. We aim to provide a center for their convergence and symbiosis at the yearly summit of the OMNeT++ community.

When looking at the technical program of the 2016 Summit, we see that the idea of an open meeting continued to exceed our expectations: researchers and developers from over 8 countries, from universities and research institutes alike, responded to our call and submitted their work. We received a wide range of contributions with propositions of new simulation models, extensions of OMNeT++ and its frameworks, tutorials and panel discussions. The discussed topics range from modeling and protocol design on all layers, application fields like routing protocols and software defined networking, to the simulation of high performance clusters and extensions of OMNeT++ for in-vehicle network architectures. We have gathered submitted contributions as well as tutorial and panel submissions of the 2016 summit in the form of proceedings to enable future reference and further discussion within the community.

We want to thank our local hosts – the Faculty of Information Technology at the Brno University of Technology and Vladimir Veselý and his team in particular – for providing us with ideal conditions to hold an event like the Community Summit. We thank our Publicity Organizers, Kyeong Soo (Joseph) Kim (Xi'an Jiaotong-Liverpool University) and Laura Marie Feeney (Uppsala University) for their excellent jobs in presenting the OMNeT++ Summit to the community. Last but certainly not least: we thank all participants of the OMNeT++ Community Summit 2016 for their work, attendance, and input. You are essential for the success of the Community Summit!

Anna Förster (University of Bremen)
Vladimir Vesely (Brno University of Technology)
Community Summit Organizers

Antonio Virdis (University of Pisa)
Michael Kirsche (BTU Cottbus-Senftenberg)
Technical Program Organizers

Contents

SESSION - ROUTING AND MOBILITY

SCTP User Message Interleaving Integration and Validation

Felix Weinrank, Irene Rüngeler, Michael Tüxen
Münster University of Applied Sciences
Dept. of Electrical Engineering and Computer Science
Bismarckstrasse 11, 48565 Steinfurt, Germany
{weinrank, i.ruengeler, tuexen}@fh-muenster.de

Erwin P. Rathgeb
University of Duisburg-Essen
Institute for Experimental Mathematics
Ellernstrasse 29, 45326 Essen, Germany
erwin.rathgeb@iem.uni-due.de

***Abstract*—The Stream Control Transmission Protocol (SCTP) is a connection and message oriented transport protocol. It supports multiple uni-directional streams in each direction allowing user message sequence preservation within each stream. This minimizes the re-sequencing delay at the receiver side in case of message loss. The base protocol, although being optimized for small messages, supports arbitrary large user messages by using fragmentation and reassembly at the cost of adding delays at the sender side. To overcome this limitation, a protocol extension called *User Message Interleaving* is currently being specified by the Internet Engineering Task Force (IETF). This paper describes the new extension, its integration and validation in the context of the INET framework.**

I. Introduction

SCTP [1] is a message oriented protocol optimized for small messages with a special emphasis on network fault tolerance. In particular, it minimizes the receiver side head-of-line-blocking by supporting multiple uni-directional streams in both directions. The sender of a user message specifies the stream being used, and sequence preservation is only guaranteed for messages sent on the same stream. Figure 1 shows the DATA-chunk used for transmitting user messages. Each DATA chunk can be identified by a transmission sequence number (TSN). Selective Acknowledgement (SACK) chunks are used to acknowledge the receipt of DATA chunks by referring to the TSNs. In case of message loss, the sender retransmits the lost DATA chunks. Therefore the TSNs are used to ensure that the receiver gets all DATA chunks sent by the sender. The stream identifier (SID) specifies the stream the user message is sent on, and the stream sequence number (SSN) is used to provide sequence preservation of user messages within each stream.

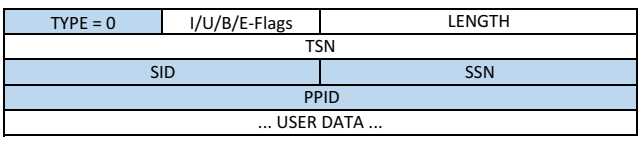


Fig. 1. DATA chunk structure

For supporting user messages larger than a packet, SCTP supports fragmentation and reassembly. The sender fragments the user message using multiple DATA chunks for transmission. In the first DATA chunk the B-bit is set in the flags field, in the last one the E-bit is set. All DATA chunks belonging to the same user message get the identical SID and SSN. The TSNs are chosen in a consecutive way. Therefore the TSN is not only used to provide reliability but also to encode the sequence of the fragments.

In most SCTP implementations, the TSNs encode the sequence in which the DATA chunks are put on the wire. Therefore, if the sender starts sending a large user message, user messages of all other streams are blocked until the sending of the large message has been completed. This introduces a sender side head-of-line-blocking.

The support of arbitrary large user messages became important in the context of WebRTC. SCTP is used as the transport protocol for WebRTC data channels, see [2]. Multiple independent data channels are multiplexed over a single peer connection. This is done by mapping data channels on SCTP streams and having a single SCTP association for a WebRTC peer connection. To avoid the sender side head-of-line-blocking introduced by fragmentation, the SCTP protocol extension to support user message interleaving [3] is specified by the IETF and required for using SCTP in the WebRTC context.

This paper is structured as follows: Section II introduces the user message interleaving extension including the new chunk types. Section III covers the required modifications to the existing stream schedulers in the SCTP simulation model. Finally, Section V show the techniques used to validate the implementation.

II. Message Interleaving

To be able to avoid the sender side head-of-line-blocking, user message interleaving uses I-DATA chunks instead of DATA chunks. When using I-DATA chunks, the TSN is only used for providing reliability. For enumerating the fragments of a large user message, the fragments sequence number (FSN) is used. For avoiding additional overhead, a single field is used to hold the payload protocol identifier (PPID) for the first fragment, when the B-bit is set. The FSN is considered 0 implicitly. For all other fragments, the FSN is set in that field. To avoid performance limitations, the 16-bit SSN is replaced by a 32-bit message identifier (MID). The I-DATA chunk is shown in Figure 2.

Parts of this work have been funded by the German Research Foundation (Deutsche Forschungsgemeinschaft).

TYPE = 64	I/U/B/E-Flags	LENGTH
TSN		
SID		RESERVED
MID		
PPID / FSN		
... USER DATA ..		

Fig. 2. I-DATA chunk structure

Furthermore, when using the partial reliability extension [4], the FORWARD-TSN chunk contains the SSN field. When using this protocol extension in combination with user message interleaving, an I-FORWARD-TSN chunk has to be used to account for the 32-bit MID. The I-FORWARD-TSN chunk is shown in Figure 3.

FORWARD-TSN chunk

TYPE = 192	FLAGS = 0x00	LENGTH
NEW CUM TSN		
STREAM 1		STREAM IDENTIFIER 1
...		
STREAM N		STREAM IDENTIFIER N

I-FORWARD-TSN chunk

TYPE = 194	FLAGS = 0x00	LENGTH
NEW CUM TSN		
STREAM 1		RESERVED
MID 1		
...		
STREAM N		RESERVED
MID N		

Fig. 3. FORWARD-TSN vs I-FORWARD-TSN chunk structure

III. Stream Scheduler

Every SCTP association has an out queue and an additional queue for every stream as illustrated in Figure 4. User data from the application is stored into the corresponding stream queue. The out queue contains the user messages which are in sending order and fragmented if necessary. Stream schedulers are used to choose the next stream queue to put data into the out queue and are configurable by the application. [3] not only specifies user message interleaving but also introduces a list of stream schedulers for SCTP. For each scheduler it is described how they operate when user message interleaving is used or not. The stream schedulers without user message interleaving support are already implemented in the existing SCTP model of INET.

Figure 4 illustrates differences between an interleaving and non-interleaving scheduler by taking the round robin scheduler as an example. When operating in non-interleaving mode the round robin scheduler chooses the first stream where the chunk zero has to be fragmented in four chunks to not exceed the maximum chunk size. The scheduler keeps locked on the first stream (S0) until all four fragments of message zero are queued in the out queue before continuing with the next stream (S1). This introduces the sender side head-of-line blocking.

In interleaving mode the scheduler chooses the first stream (S0) and iterates over all streams chunk by chunk regardless of fragmentation.

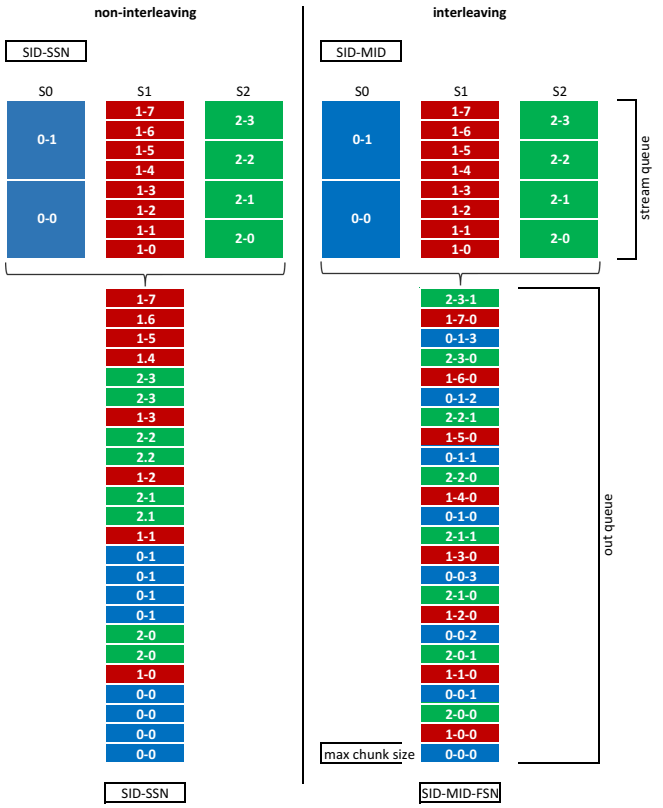


Fig. 4. Message queueing with and without interleaving by a round robin scheduler

IV. Implementation

The implementation of the SCTP message interleaving extension has been done in two steps in the INET source code.

In a first step, we extended the SCTP model itself by adding the new chunk types, interleaving logic at the sender side and extended the existing stream schedulers with interleaving support.

To configure the interleaving extension we added the `iData` parameter to the SCTP model. When the `iData` parameter is set to `true` the SCTP model offers message interleaving support in the 4-way-handshake to its peer. If both peers offer interleaving support, the I-DATA and I-FORWARD-TSN chunks are used instead of the DATA and FORWARD-TSN chunks, and the SCTP model operates in interleaving mode.

Whenever receiving an I-DATA chunk SCTP iterates over the receiver queue to determine if it can successfully reassemble the data message by comparing MIDs and FSNs.

In the second step, we added support for the external interface and PCAP recorder files by adding the chunk types to the serializer and deserializer modules of INET. We will address this topic in detail in the validation section.

As mentioned in Section III, the SCTP model already includes all stream schedulers except the weighted fair queueing scheduler, but all schedulers lack the interleaving support.

V. Validation

To ensure the correct implementation of the interleaving extension, we used multiple techniques to validate the model operation within the INET framework (3.4) of the OMNet++ (5.0) simulation environment.

A. Manual Packet Flow Inspection using Wireshark

We were able to use Wireshark (2.0.2) to analyze the recorded PCAP files, since we implemented support for the SCTP interleaving extension earlier. This helped us a lot to validate the correct message flow and find bugs in our implementation.

B. Interoperability Testing

After we successfully validated the model with Wireshark we ran tests with the SCTP implementation of the FreeBSD (11-current) kernel via the external interface. We focused on the interleaving negotiation in the 4-way-handshake and the data transfer using I-DATA chunks. Bulk transfer with multiple message loss rates was used and no issues found.

C. Automated Packet Flow Inspection using Packetdrill

Packetdrill [5] is a script-based testing tool for transport protocols on Unix-based operating systems which has been released in 2013 by Google. The great advantage of Packetdrill is the testing of specific scenarios like the usage of wrong parameters, which is not possible in usual test setups. To benefit from this advantage, we ported Packetdrill to INET [6]. It is already part of the current version and supports UDP, TCP and SCTP.

To be able to test the I-DATA chunk, Packetdrill had to be extended by the new chunk type, the new parameters and code to create new chunks to compare them to the ones sent by the SCTP stack. As the interleaving support is negotiated in the handshake, the new socket options had to be added to trigger the `iData` parameter mentioned in section IV. Our goal to run the same test scripts as the kernel implementation was achieved by porting the use of `ifdef` clauses from the kernel version. Thus, the script can be tailored to handle OS specific features. An example are error causes, that are not part of the simulation.

As the SCTP I-DATA feature was already integrated in Packetdrill for kernel SCTP, there were about 100 tests that could be applied to INET. Several of those tests focus on the appropriate handling of received packets resulting from an inappropriate behavior of the sender. While testing this for a real implementation is important, simulation might assume appropriate behavior of the sender. Therefore, these test would not be passed by the simulation and have been disabled. Nevertheless, more than 80 tests were passed that focussed on the conformance with the protocol specification. These tests cover the negotiation of the extensions, the correct handling of the TSNs, and sending and receiving of fragmented messages.

D. Measurements

To measure the improvements by message interleaving we built a typical WebRTC data channel scenario with two competing streams within one association.

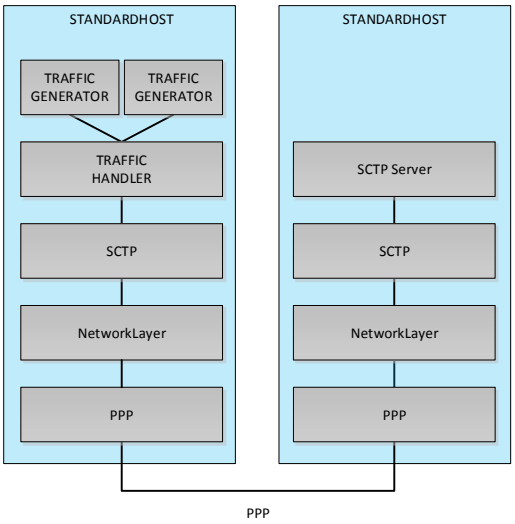


Fig. 5. Traffic generator and handler architecture

To have an adjustable testbed, we created a traffic generator module and a traffic handler module as shown in Figure 5. Each generator represents a single SCTP stream and is connected to a handler which establishes the connection to a remote peer.

While the first stream has a low priority and is saturated by large messages the second stream sends small messages with a high priority. This is a common scenario for WebRTC applications like a chat program with file transfer capabilities.

While keeping the small messages between 8 - 16 bytes we increased the message size of the larger messages from 4 kilobytes to 128 kilobytes. We were mainly interested in the message end-to-end delay.

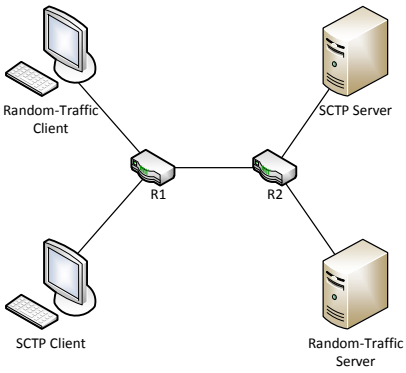


Fig. 6. Bottleneck scenario with SCTP and background traffic

The traffic generators are intentionally separated from the traffic handler to support different generators with the same handler or the same generator for several protocols. Figure 7 shows the configuration of a simple traffic generator as a saturated sender with an increasing message size for every run. The traffic handler coordinates the generators and communicates with the SCTP module. On the receiver side, we modified the SCTPServer to record the message delay not only per association but also for every stream.

```
# settings for client - first generator
...gen[0].typename = "TrafficgenSimple"
...gen[0].name = "low prio"
...gen[0].id = 1
...gen[0].priority = 128
...gen[0].packetCount = -1
...gen[0].packetSize = ${ps=4 .. 128 step 2}kB
...gen[0].packetInterval = 0ms
...gen[0].startTime = 5s
...gen[0].stopTime = 65s
```

Fig. 7. Settings for a saturated traffic generator

To achieve a more realistic testing scenario and get some randomness, we used a scenario where the SCTP traffic competed with a random UDP sender which sent a small amount of traffic - see Figure 6.

While increasing the message size of the saturated low priority stream we expected a linearly increasing delay for the high priority stream in non-interleaving mode.

For the interleaving mode scenario, we expected the high priority stream delay at a constant level. As shown in Figure 8, the delay for non-interleaving mode rises constantly while staying on a constant level when operating with interleaving.

The theoretical message delay $d_n(s_{\text{msg}})$ in the non-interleaving mode and $d_i(s_{\text{msg}})$ in the interleaving mode depends on the user message size s_{msg} and is calculated by adding the link delay d_{link}, the delay caused by the link buffers d_{buffer} and the specific delay caused by the SCTP buffers which is different for interleaving and non-interleaving mode.

For computing the specific delay in non-interleaving mode, we take the user message size s_{msg} and calculate how many SCTP packets are required to transfer the user message by dividing the user message size by the maximum fragment size s_{frag}. The overhead is the product of this number and the corresponding header size s_{hdr} for SCTP, IP and PPP. Adding the user message size to the overhead and dividing by two gives the average buffered head-of-line-blocking size. Finally dividing by the available bandwidth bw gives the specific delay in the non-interleaving mode. For the interleaving mode the specific delay is calculated by dividing the maximum transmission unit mtu, divided by 2, by the link bandwidth.

$$d_n(s_{\text{msg}}) = \frac{s_{\text{msg}} + s_{\text{hdr}} \cdot \left\lceil \frac{s_{\text{msg}}}{s_{\text{frag}}} \right\rceil}{2 \cdot \text{bw}} + d_{\text{link}} + d_{\text{buffer}}$$

$$d_i(s_{\text{msg}}) = \frac{\text{mtu}}{2 \cdot \text{bw}} + d_{\text{link}} + d_{\text{buffer}}$$

Figure 8 shows a comparison between our expected results and the results from our simulation which meets our expectations.

VI. Conclusion and Outlook

The message interleaving extension for SCTP solves head-of-line-blocking for fragmented messages in SCTP. We integrated the extension into the SCTP model, verified its correct

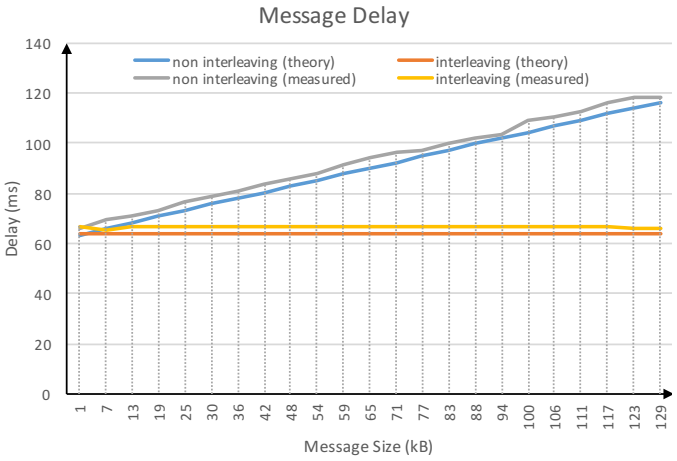


Fig. 8. Message delay with and without interleaving

implementation by internal tests with Packetdrill as well as with interoperability tests with the real implementation in FreeBSD. In our ongoing work, we will focus on buffer optimization and additional stream schedulers like the weighted fair queueing scheduler. We will contribute the interleaving extension and the extended stream scheduler to the INET framework.

VII. Acknowledgements

We would like to thank Julius Flohr from the University of Duisburg-Essen for the cooperation on the generator module.

References

[1] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, updated by RFCs 6096, 6335, 7053. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt

[2] R. Jesup, S. Loreto, and M. Tuexen, "WebRTC Data Channels," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-rtcweb-data-channel-13, January 2015. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-rtcweb-data-channel-13.txt

[3] R. Stewart, M. Tuexen, S. Loreto, and R. Seggelmann, "Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-tsvwg-sctp-ndata-06, July 2016. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-sctp-ndata-06.txt

[4] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension," RFC 3758 (Proposed Standard), Internet Engineering Task Force, May 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3758.txt

[5] N. Cardwell, Y. Cheng, L. Brakmo, M. Mathis, B. Raghavan, N. Dukkipati, H.-k. J. Chu, A. Terzis, and T. Herbert, "packetdrill: Scriptable Network Stack Testing, from Sockets to Packets." in *USENIX Annual Technical Conference*, 2013, pp. 213–218.

[6] I. Rüngeler and M. Tüxen, "Integration of the Packetdrill Testing Tool in INET," *CoRR*, vol. abs/1509.03127, 2015. [Online]. Available: http://arxiv.org/abs/1509.03127

Enabling Soft Vertical Handover for MIPv6 in OMNeT++

Atheer Al-Rubaye, Ariel Aguirre, Jochen Seitz
Communication Networks Group
Technische Universität Ilmenau, Germany
{atheer.al-rubaye, ariel.aguirre, jochen.seitz}@tu-ilmenau.de

***Abstract*—Switching connectivity over multiple wireless interfaces of a mobile node is essential when performing handover between heterogeneous networks. In such a communication scenario, session continuity as experienced by the user can be enhanced if soft handover is enabled through the concept of make-before-break. However, some of the available networking protocols need to be modified to support this feature. This work gives an insight into the supplementary modules we implemented and the modification conducted in MIPv6 model of OMNeT++ to facilitate soft handover and data offloading for mobile nodes.**

***Index Terms*—Soft Handover; MIPv6; Link Layer Controller.**

I. Introduction

Nowadays, heterogeneous communication networks coexist and overlap over areas so frequently. Considering the increasing availability of powerful phones equipped with multiple interfaces for different technologies, it can be employed to switch connectivity in between through a process known as vertical handover (VHO). A simple VHO scenario is start moving from home while having an active stream running, passing through the city, and reaching office, as illustrated in figure 1. However, to offer a more sophisticated handover (HO), session continuity is a major factor to consider. A widely implemented protocol to handle the identification of mobile nodes upon roaming between networks of different subnet address (layer 3 HO) is mobile IP in its two versions 4 and 6 [1], [2].

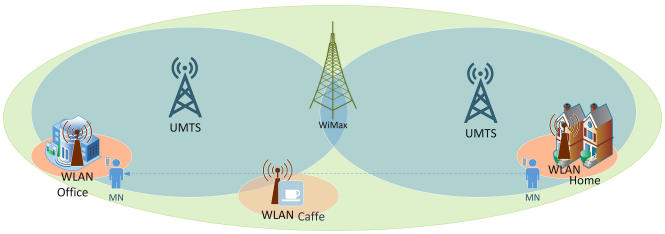


Fig. 1. Vertical Handover

In this work, we give an insight into the modules and procedures we developed to add support of multiple interfaces utilization and provide session continuity to the solutions that are based on mobile IP in OMNeT++. Decision making and address resolution phases of handover are out of the scope here, where the main focus is on the management of interfaces connectivity. The concept of make-before-break is therefore implemented to provide a soft switching of traffic between the available wireless interfaces/networks. An extension to our previous work in [3] is presented to support mobile IP version 6 (MIPv6) and adopt its functionality in xMIPv6 model [4] of INET v2.6 framework in OMNeT++ [5]. The support of the availability of multiple wireless interfaces with possible connectivity switching in between motivates further investigations and, hence, modifications to implemented protocols and modules. [3] implements a link layer controller module (vhoCtrler) within a compound model named IPCoManager to support soft handover in IPv4-based MNs. IPCoManager includes also handover decision algorithms in a more sophisticated form and an agent module that manages address resolution, as explained in [3], these are however out of focus here.

To enable this functionality for MIPv6 modules in INET, a set of modifications and adaptations are necessary to be conducted in xMIPv6 and in the vhoCtrler as well. Some are related to overcoming some limitations in xMIPv6, to enable more realistic mobility scenarios, like to have many handovers between the home and foreign networks when moving back and forth in between, and some are related to how IP address assignment and update mechanism should be carried out with the existence of more than one wireless interface. Enhancements in handover in terms of QoS parameters are investigated through simulations that emphasize the difference between a hard and a soft handover, and their impact on running sessions.

This work is still under development and related modules may need further enhancements and modifications therefore, the code will be available after a successful finish of the PhD research work of the correspondent author.

The rest of the paper is organized as follows: section II gives an overview of MIPv6, in section III, an insight into the implemented and modified modules to support soft switching between interfaces is given, section IV describes the simulation scenarios and discusses the measurements, finally, the work is concluded in section V.

II. Overview of MIPv6

The protocol MIPv6 enables an IPv6 mobile node to be identified and reachable at any time, regardless to its point of attachment to the Internet. In MIPv6, an MN is always identified through a fixed address called Home Address (HoA), which is derived from the home network IPv6 prefix. Once it is visiting a foreign network, the MN is reachable through a

Care-of-Address (CoA), which is formed based on the prefix announced in that network. At any time, the anchor point of the MN is an entity in the home network, which the standard names Home Agent (HA). MIPv6 enables the MN and HA to keep a binding of the CoA and HoA. Each time a MN changes its point of attachment to the Internet, it notifies the HA to update the binding with a new CoA. In such a way, packets sent by a correspondent node (CN) to the MN, are sent to the home network, intercepted by the HA, which then forwards them to the MN's CoA. In the reverse, packets sent by the MN to the CN are routed directly to it if the firewall in between permit [1].

III. Implementation of Management Concept

A. Management at the Link Layer

The controller module in [3] is a controlling entity to all wireless interfaces at the link layer. It receives requests from the underlaying interfaces for permissions to associate to its relevant networks when available in range.

Regardless of the interface type, its highest management entity must be modified to acquire a permission before associating to its corresponding network. For example, the Agent module inside the wireless Ieee80211 compound module of INET will need permission from the vhoCtrler before allowing underlaying modules to associate to the related access point. It may also receive a command to disconnect an association with an access point if a handover decision is made and a switching process has taken place in favor of another interface.

The controller may consult a decision module for a selection, which compares between the current connected interface/network and the requesting one. However, the decision criteria and algorithm implementation are out of the scope here. If a request is permitted, vhoCtrler releases the old connection, but only when the new one is confirmed to be associated, configured and hence, ready to undertake traffic.

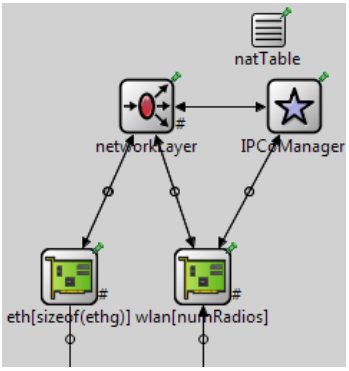


Fig. 2. Management of Interfaces

To hold/retrieve the identities and parameters of multiple interfaces, the vhoCtrler module employs a structure for interface/network attributes. Three objects are defined accordingly, these namely are; serving, previous and candidate networks/interfaces for currently connected, previous and under examination interfaces respectively. The identities include the interface ID and the associated IP address. The parameters include attributes of the related access point like the received signal strength, the signal to noise ratio, and some other that might be transmitted by the network service provider like cost of service and limit of supported mobility speed. It includes also context information of an interface like the available credit (in data volume or monitory units) and hence, the expected bit rate of the possible multiple networks associations. The parameters are updated periodically with each received beacon/advertisement in relevance to each interface, and during the application run to enable a sophisticated decision making, which might be based on multiple attributes.

It is important to notify here that the vhoCtrler does not deal with data traffic, but only exchanges messages with the aforementioned modules for purpose of management. Figure 2 illustrates the implemented module within the TCP/IP model of INET.

B. Modifications at the Network Layer

The need for modifications on the network layer in general, and the xMIPv6 in specific were necessary, because the availability of and the switching between multiple wireless interfaces was not tested before, and due to some missing functionality while employing a mobility model beyond the one implemented in the INET example of xMIPv6.

When a mobile node associates to a new wireless network, it starts receiving router advertisements (RA) from the router in that network. The process on the received RA packets in the neighbor discovery procedure needed to be modified in order to make home-relevant information available to the serving interface, which is achieved by adding two pieces of information to the related serving wireless interface in the interface table. The first is the home network information and the second is the home address, but in tentative mode. When the xMIPv6 class need to employ the addresses of the home agent and home address when a binding update (BU) is required to be sent, it queries the vhoCtrler to identify the interface that is labeled as the serving one momently, and retrieve the home information from the IPv6InterfaceData class accordingly.

Upon a handover, the routing table needed to be updated by deleting routing information related to the previous interface so, before the mobile node may employ its IPv6 address in the new network, the duplicated address detection (DAD) process is invoked. Once the scope of this address changes to global, a notification of this change is received by the vhoCtrler, which updates then the mobile node's routing table by removing the default route and prefix associated with the previous interface. This assures that the BU will be routed through the newly connected interface using a topologically correct address. The MIPv6 registration process follows as described in standard [1]. For the case when an MN initiates a communication session while away from home, we have updated the original IPv6 procedures. Now, as in this case the MN and the home agent have built a bidirectional tunnel between them, we have filled the source IP address of the inner datagram with its

home address, as described in the protocol standard in [1]. This update is required in order to enable the corresponding node (CN) to return packets to the mobile node through the Home Network.

IV. Simulations and Measurements

In our simulated scenario, we consider a topology of two networks, one acts as the home network and the other as the foreign network in a free space environment. The access point of each is configured with the same transmission power but, on two different channels. The mobile node starts a video/VOIP traffic with the CN and moves in the meanwhile between the two networks to perform several handovers. As mobility model, we employ the TractorMobility provided in INET, which provides a path similar to a tractor moving through a field with specific number of rows, as illustrated in figure 3.

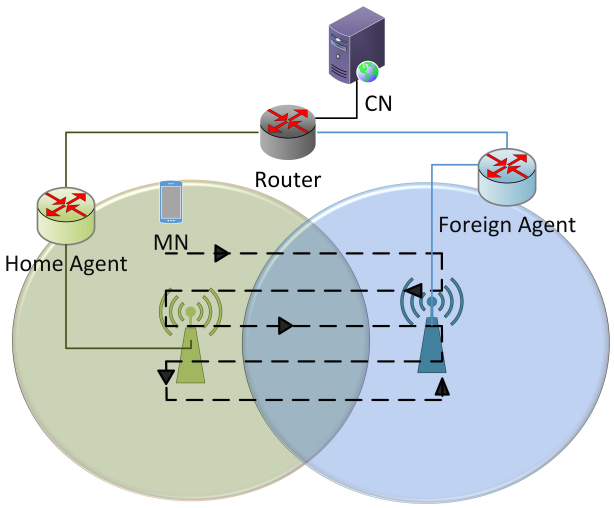


Fig. 3. Simulation Network Topology

Two cases are considered, one with the MN using the basic version of MIPv6 of OMNeT++, which we refer to as Hard HO, and the second using our vhoCtrler module as well, which we refer to as Soft HO. The MN is equipped with a single wireless interface (Wi-Fi IEEE 802.11 with 2 Mbps bit rate) for the first case while, with two interfaces for the second case. In this case, each interface is configured to associate to only to one of the networks in the topology such that, the first interface connects only with the access point of the home network while, the second connects only with the access point of the foreign network. Roaming therefore requires switching of the traffic over the two interfaces in this scenario. Two real-time applications are employed; video stream and voice over IP. Sending rates of 0.5 and 2 Mbps were used for the video traffic, while a 5 ms play out delay and 20 ms packetization interval for the VOIP traffic.

The simulation was repeated five times with the same seed, using different mobility speed values in each (1, 2, 4, 8 and 10 mps). The simulation time differs for each run (2000 to 200 seconds) in proportion to the mobility speed in order to have the same number of handovers (10 times) in each run. The effect of the handover process is observed on the packet loss rate, which is a major parameter for real-time applications.

Figures 4 and 5 illustrate the packet loss rate (for the video and VOIP traffic respectively). We notice a significant enhancement presented by soft handover when enabled for MIPv6, especially at higher velocities. However, for the VOIP application, the packets are not sent as a stream, but as talk spurs therefore, lately arrived packets might be also considered as lost in this case.

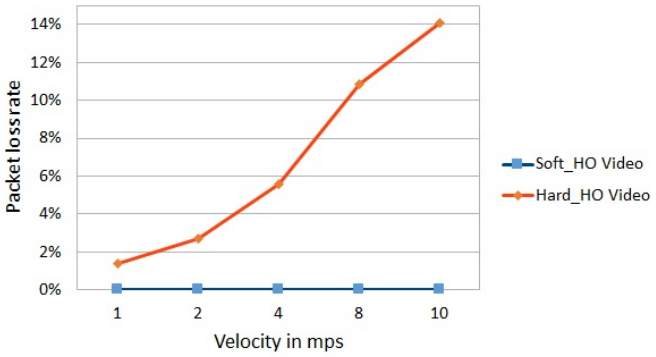


Fig. 4. Packet loss rate in the Video traffic case

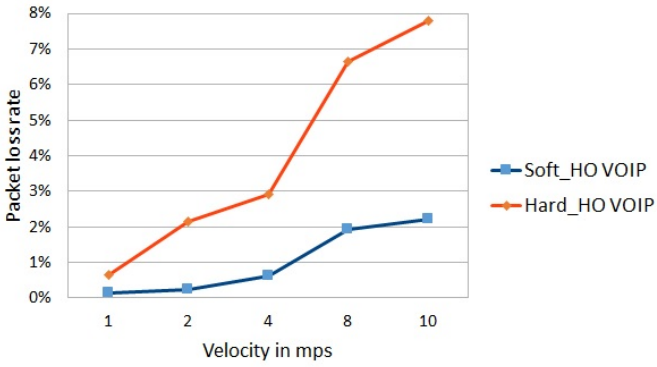


Fig. 5. Packet loss rate in the VOIP traffic case

The Mean Openion Score (MOS) measured by OMNeT++ for the VOIP application shows relatively similar performance at low speeds for both HO schemes but, better values can be noticed in higher speeds for the Soft HO as compared to the hard one, as can been seen in figure 6.

Handover in Soft HO case takes place while being in the coverage overlap area between the two networks and switching is executed only when the second connection is really created (make-before-connect). In Hard HO case, handover takes place only when the MN is on the edge of the connected network, where it starts associating its single wireless interface to the upcoming network after notifying the absence in the received beacons of the connected one.

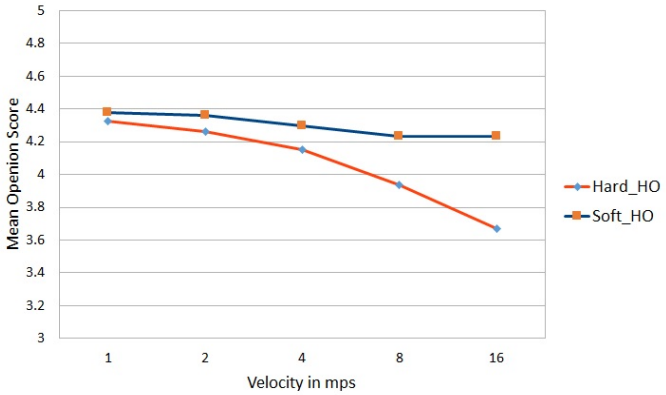


Fig. 6. MOS of the VOIP traffic case

V. Conclusions and Future Work

To enable soft handover in the MIPv6 model implemented in INET/OMNeT++, we needed to adopt the xMIPv6 model and our controlling entity as well, which we implemented in an earlier work of us for IPv4-based models. A set of modifications to the xMIPv6 were necessary to support the availability of multiple interfaces and to enable a correct operation of the protocol according to the standard. In the two tested applications, a soft switching between two interfaces in a coverage overlap area has highly reduced the number of lost packets upon a handover. However, an overall examination of the code in comparison to the xMIPv6 model is still necessary to ensure compatibility with the protocol standard and a correct functionality within more sophisticated simulation scenarios. Adding features of extended versions of MIPv6, like HMIPv6 and PMIPv6 to xMIPv6 can be considered as a significant contribution to the OMNeT++ community as a future work.

References

[1] D. Johnson and C. Perkins, "Mobility Support in IPv6," in *RFC 3775*, 2004. [Online]. Available: https://tools.ietf.org/html/rfc3775

[2] E. Perkins, "IP Mobility Support for IPv4," in *RFC 3344*, 2002. [Online]. Available: https://www.ietf.org/rfc/rfc3344.txt

[3] A. Al-Rubaye and J. Seitz, "Dynamic Index NAT as a Mobility Solution in OMNeT++," in *Proceedings of the OMNeT++ Community Summit*, 2015.

[4] F. Yousaf, C. Bauer, and C. Wietfeld, "An accurate and extensible mobile IPv6 (xMIPV6) simulation model for OMNeT++," in *1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2008.

[5] A. Varga and O. Ltd., *OMNeT++ User Manual version 4.3*. [Online]. Available: http://www.omnetpp.org/

Performance and Security Evaluation of SDN Networks in OMNeT++/INET

Marco Tiloca
SICS Swedish ICT AB, Security Lab
Isafjordsgatan 22, Kista (Sweden)
Email: marco@sics.se

Alexandra Stagkopoulou
KTH Royal Institute of Technology
Isafjordsgatan 22, Kista (Sweden)
Email: stagk@kth.se

Gianluca Dini
University of Pisa
Largo Lazzarino 1, Pisa (Italy)
Email: gianluca.dini@unipi.it

***Abstract*—Software Defined Networking (SDN) has been recently introduced as a new communication paradigm in computer networks. By separating the control plane from the data plane and entrusting packet forwarding to straightforward switches, SDN makes it possible to deploy and run networks which are more flexible to manage and easier to configure. This paper describes a set of extensions for the INET framework, which allow researchers and network designers to simulate SDN architectures and evaluate their performance and security at design time. Together with performance evaluation and design optimization of SDN networks, our extensions enable the simulation of SDN-based anomaly detection and mitigation techniques, as well as the quantitative evaluation of cyber-physical attacks and their impact on the network and application. This work is an ongoing research activity, and we plan to propose it for an official contribution to the INET framework.**

***Index Terms*—SDN; Security; OMNeT++; INET; Simulation**

I. Introduction

In the recent years, *Software Defined Networking* (SDN) [10] has been more and more adopted as a new network communication paradigm [3]. Unlike the traditional Internet model, SDN separates the actual forwarding of network packets (data plane) from the management of network traffic and routes (control plane). In principle, a centralized *SDN controller* determines how to handle different traffic segments, namely *flows*, and installs related forwarding rules on simple *switch* devices responsible for the actual packet forwarding. The SDN controller and the switches rely on a common set of APIs and control messages to interact with each other, so preventing interoperability issues among devices from different vendors. To this end, *OpenFlow* [4] has become the de-facto protocol implemented in SDN controllers and switches. By entrusting all the monitoring and decision processes to the SDN controller, SDN considerably simplifies the management of large-scale networks. Also, it results in a faster and more flexible re-configuration of traffic patterns, if compared with traditional network architectures.

To deploy SDN networks that operate according to expectations, it is vital that, far before deployment, network designers can *quantitatively* evaluate: i) network and communication performance; ii) effects and impact of security attacks against the network; and iii) accuracy and effectiveness of anomaly detection systems [6]. To this end, network simulation represents a convenient and helpful tool to adopt, since it can be infeasible to perform the same evaluations in real, large-scale, networks. Especially for evaluating SDN-based monitoring systems and the impact of security attacks, it is convenient to perform such assessments at design time, so not interfering with the regular operations of real networks. On the other hand, an analytical approach is often infeasible, unless oversimplifying assumptions are made.

So far, there have been only a few contributions for simulation of SDN networks. Mininet is a common tool to perform functional testing of emulated networks based on OpenFlow [1]. However, it focuses on real-time functional testing rather than on the simulation and evaluation of arbitrary network scenarios. The network simulator NS-3 provides an OpenFlow simulation model [9]. However, the SDN controller is not modeled as an external entity, and thus it is not possible to quantitatively evaluate the impact of the control channel or to consider multiple switches connected to the same SDN controller. More recently, [2] has proposed the implementation of OpenFlow components integrated in the INET framework [5], based on the network simulation environment OMNeT++ [15]. However, it models only the basic flow establishment between OpenFlow switches and a basic SDN controller. Also, it implements only flow-matching rules solely based on MAC address fields.

In this paper, we describe a set of extensions for the INET framework that enable performance and security evaluation in SDN networks. In particular, we have extended the model initially proposed in [2], in order to support additional OpenFlow messages and enable the processing of network packets based on flow-matching rules of arbitrary complexity. Also, we have provided support for SDN-based monitoring systems, according to which the SDN controller: i) collects flow statistics from the connected switches; ii) analyzes collected samples in order to detect possible anomalies; and iii) possibly determines and disseminates policies to mitigate anomalies and restore normal operating conditions in the network. Finally, we have enabled the simulation of effects of security attacks in SDN network scenarios. To this end, we consider the INET-based attack simulation framework SEA++ that we previously described in [8] and whose functionalities have been adapted to support attack simulation in SDN networks. Intuitively, the user can describe different cyber-physical attacks by means of a high-level attack specification language, without altering the actual

implementation of any of the INET software components. Events that reproduce the effects of the described attacks are injected at runtime during the simulation experiments. The approach devoted to reproducing and evaluating security attacks is not strictly related to SDN, i.e. it can in principle be reused together with any network architecture and scenario sopported by the INET framework.

We believe that our extended simulation tool allows researchers and network designers to effectively and conveniently evaluate SDN architectures at design time, both in attack-free scenarios and in case different security attacks are performed. In particular, it makes it possible to evaluate an SDN scenario in terms of: i) network and communication performance in an attack-free case; ii) effectiveness and reactiveness of SDN-based monitoring systems; iii) quantitative effects of security attacks, as to how attacks affect performance indicators in the same network scenario; and iv) quantitative effectiveness of security countermeasures. This work is an ongoing research activity, and we plan to propose it for an official contribution to the INET framework. Our extended simulation framework is currently under development, and the source code is available at [7].

The paper is organized as follows. Section II overviews Software Defined Networking. Section III describes our extensions to the INET framework which support SDN, OpenFlow, SDN-based monitoring systems, and simulation of effects of security attacks. In Section IV, we evaluate a simple Denial of Service attack against a server host, and present preliminary results. Finally, Section V concludes the paper and anticipates future works and research directions.

II. Packet forwarding in SDN

SDN essentially relies on the separation of *data plane* and *control plane*. In particular, the data plane is entrusted to simple *switches* that forward network packets according to stored *flows* and related matching rules. The control plane is entrusted to a centralized SDN *controller*, which establishes packet flows and installs them on the switches. The SDN controller and switches interact with each other through dedicated control messages and APIs, such as the ones provided by the common *OpenFlow* protocol [4].

Figure 1 shows an example of flow establishment and packet delivery. First, host A sends a packet P to host B (step 1). Upon receiving packet P, the switch looks for a possible match between P and the flows installed in its flow table (step 2). If no matching rule to process packet P is found, the switch asks the SDN controller for further instructions (step 3). Then, the SDN controller creates a new flow (step 4), and installs the related matching rule and actions on the switch (step 5). That is, it instructs the switch that all packets coming from Port 1, with IP source address 192.168.0.1 and IP destination address 192.168.0.2 must be sent out over Port 2. After that, the switch installs the new flow on its flow table, as a pair of matching fields and actions to be performend on any packet matching with that flow (step 6). If more output ports are available, the switch is initially instructed to send out this first packet P over

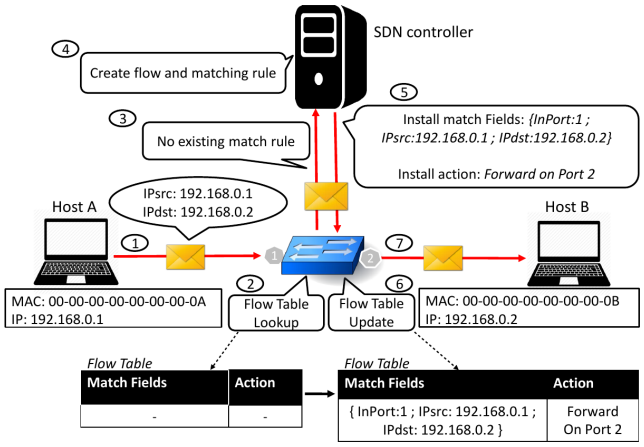


Fig. 1: Flow establishment and packet forwarding.

all ports different than Port 1. Later on, upon receiving a reply packet on a specific port, the switch contacts again the SDN controller, which modifies the action associated to that flow by specifying the exact outgoing port to consider. Finally, the switch forwards packet P to host B (step 7).

III. SDN extensions for INET

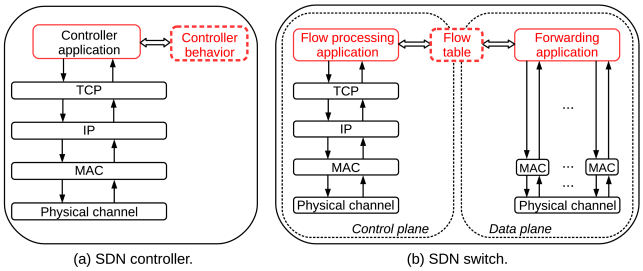
This section overviews our extensions for the INET framework. Section III-A presents the support for flow establishment and packet forwarding based on OpenFlow. Section III-B presents the support for SDN-based monitoring systems. Section III-C presents the support for the evaluation of security attacks. Our extended simulation framework is under development, and the source code is available at [7].

A. Support to SDN architecture and OpenFlow

SDN relies on two fundamental elements: i) the SDN controller and switches; and ii) the exchange of OpenFlow messages to establish flows and install them on the switches.

We have considered the model initially proposed in [2], that provides a number of essential OpenFlow messages and the implementation of the switches and SDN controller nodes. In particular, the SDN controller is essentially a host running an application which relies on a typical TCP/IP stack and models a specific controller behavior. Instead, the switches are modelled as a new type of node, where a control plane running a TCP application on top of a TCP/IP stack interacts with multiple data plane instances, by means of the OMNeT++ signal concept. Both the SDN controller and the switches rely on a specific time model to take into account the processing time of real OpenFlow units.

We extended the model implemented in [2], providing additional functionalities that enable the evaluation of SDN-based monitoring systems and impact of security attacks (see Sections III-B and III-C). In particular, we provided additional support to: i) exchange and process the OpenFlow control messages OFPT_STATS_REQUEST and OFPT_STATS_REPLY between the SDN controller and the switches for collection of flow statistics; ii) exchange and process the OpenFlow



(a) SDN controller. (b) SDN switch.

Fig. 2: Overview of a SDN controller and switch.

messages OFPT_FLOW_REMOVED sent by the switches to the SDN controller to report expired flows; and iii) allow the switches to perform the matching of incoming packets with installed flows based on arbitrary packet fields (rather than on MAC addresses only).

Figure 2 shows the architectural overview of the SDN controller (a) and a SDN switch (b). The elements coloured in red have been extended in our implementation, in order to enable statistic collection and network monitoring, as well as the reporting of flow expiration and the arbitrary-complex matching of packets with flows installed on switches. The SDN controller is simply modelled as a generic host, running a traditional TCP/IP stack. Then, a specific *Controller application* is responsible for the establishment of flows, and their installation, update and revocation on the switches. Of course, the *Controller application* can be entrusted with additional services, such as network monitoring and anomaly detection (see Section III-B). Policies, algorithms and parameters according to which the *Controller application* behaves are specified in the *Controller behavior* module.

The switch is composed of two different segments, sharing the same *Flow table*. That is, the *Control plane* is also modelled as a typical TPC/IP stack, through which a *Flow processing application* can exchange control messages with the SDN controller. Instead, the *Data plane* is a set of minimal communication stacks, each one relying on a dedicated MAC interface. Then, all MAC interfaces are connected to the same *Forwarding application*, which forwards packets from an incoming MAC interface to an outgoing MAC interface, according to the flow-matching rules and related forwarding actions stored in the *Flow table*. If no matching is produced, the *Forwarding application* asks the *Flow processing application* to contact the SDN controller and establish a new flow, before proceeding.

The following OpenFlow messages are considered in order to support the establishment and updates of flows.

- OFPT_PACKET_IN. Sent by the switch to the SDN controller, when a packet is received and no match is produced.
- OFPT_PACKET_OUT. Sent by the SDN controller to a switch, specifying to send a packet over a specific interface.
- OFPT_FLOW_MOD. Sent by the SDN controller to a switch, specifying to install/modify a flow in its flow table.
- OFPT_FLOW_REMOVED. Sent by a switch to the SDN controller, notifying that a flow in the flow table has expired.

B. SDN-based monitoring systems

The SDN controller can run additional application services to perform security monitoring of the network. This practically relies on three modules, namely *flow statistic collection*, *anomaly detection*, and *anomaly mitigation*. That is, the SDN controller periodically sends an OFPT_STATS_REQUEST OpenFlow message to the switches, according to a pre-configured polling interval. The switches reply with an OFPT_STATS_REPLY OpenFlow message, reporting the packet matches and the accesses to their flow tables occurred during the current time window. Given this information, the SDN controller can analyze the collected statistics, and look for possible anomalies or ongoing attacks, such as Denial of Service (DoS) or wormhole propagation. The actual anomaly detection process can rely on several different techniques, e.g. machine learning [14], data mining [13], or entropy based algorithms [11] [12].

When the SDN controller identifies traffic anomalies or ongoing attacks, it performs mitigating actions to limit or neutralize their impact. That is, the SDN controller sends OFPT_FLOW_MOD messages to specific switches, to install or update flows in their flow tables. Such flows and related policies aim at blocking malicious traffic, e.g. by dropping or caching packets that are addressed to presumed victim hosts or coming from suspected attack sources.

C. Simulation of security attacks

Our extensions allow network designers to *quantitatively* evaluate the impact and effects of security attacks against SDN networks, i.e. how attacks affect performance indicators with respect to the same scenario in the attack-free case. This makes it possible to rank different attacks according to their severity, and hence to easier select effective countermeasures to adopt. Rather than executing security attacks by implementing their actual performance, we *reproduce* their effects against the network and applications. Evaluation of security attacks relies on two fundamental components, i.e. a high-level *Attack Specification Language* and an *Attack Simulation Engine*, described in Sections III-C1-III-C3.

We previously presented an earlier version of such components as part of the INET-based attack simulation framework SEA++ [8], and adapted their functionalities to support attack simulation in SDN networks. Note that the approach adopted to reproduce and evaluate security attacks is not strictly related to SDN, i.e. it can in principle be reused for any network architecture sopported by INET. At the same time, our extended simulation tool makes it possible to evaluate the impact of security attacks that specifically consider switches as actual attack victims (e.g. injection of fake flows to install) or compromised units contributing to the attack execution (e.g. through packet dropping or replication). Our implementation activity is currently focused on enabling the evaluation of such attacks involving switches.

1) Attack description: The Attack Specification Language (ASL) allows the user to describe attacks to be evaluated, in terms of their final *effects*. That is, the user assumes that

attacks can be successfully performed, regardless how an adversary can specifically mount and execute them. Then, the user describes attacks as sequence of events that atomically take place during the network simulation. To this end, the ASL provides a collection of *primitives* organized into two sets, i.e. *node primitives* and *message primitives*.

Node primitives account for *physical attacks* against network nodes. A physical attack is composed by a single node primitive. The following node primitives are available:

- destroy(nodeID, t) - Remove node 'nodeID' from the network at time 't', after which it cannot take part to network communication any longer.
- move(nodeID, t, x, y, z) - Change the current position of node 'nodeID' to a new position {x,y,z} at time 't'.

Message primitives account for *cyber attacks* and describe actions on network packets. Packet fields are addressed by means of the *dot* notation packet.layer.field. The following node primitives are available:

- drop(pkt) - Discards the packet 'pkt'.
- create(pkt, fld, content, ...) - Creates a new packet 'pkt' and fill its field 'fld' with 'content'. It is possible to specify the content of multiple fields through a single invocation.
- clone(srcPkt, dstPkt) - Produces a perfect copy 'dstPkt' of the packet 'srcPkt'.
- change(pkt, fld, newContent) - Writes 'newContent' into the field 'fld' of packet 'pkt'.
- send(pkt, d) - Schedules the transmission of a packet 'pkt' produced by 'clone()' or 'create()', after a delay 'd'.
- retrieve(pkt, fld, var) - Assigns the content of the field 'fld' of packet 'pkt' to the variable 'var'.
- put(pkt, dstNodes, TX | RX, updateStats, d) - Inserts the packet 'pkt' either in the TX or RX buffer of all nodes in the 'dstNodes' list, after a delay 'd'.

The ASL provides statements to specify *conditional attacks*, i.e. lists of events described through message primitives that occur on a declared list of nodes if a condition is evaluated as TRUE. That is, as a general example:

```
from T nodes = <list of nodes> do {
  filter(<condition>) <list of events>
}
```

Also, it is possible to specify *unconditional attacks*, i.e. list of attack events described through message primitives, and reproduced on a periodical fashion or upon the occurrence of specific conditions evaluated by network nodes at runtime. For instance, the statement `from T every P do {<list of events>}` specifies that the list of events takes place periodically on the declared list of nodes, since time T and with period P.

2) Attack Simulation Engine: After having described the attacks to be evaluated, the user simply runs a simulation campaign on the enhanced INET framework, in order to evaluate the impact and effects of the described attacks. To this end, the *Attack Simulation Engine* (ASE) considers network nodes as implemented by an *Enchanced Network Node* module. The latter is in turn composed of: i) an *Application* module possibly including different sub-modules modelling the actual node application(s); ii) an arbitrarily complex collection of protocols composing the communication stack; and, finally iii) a *Local Event Processor* (LEP) module. Notice that all such modules but LEP can be off-the-shelf.

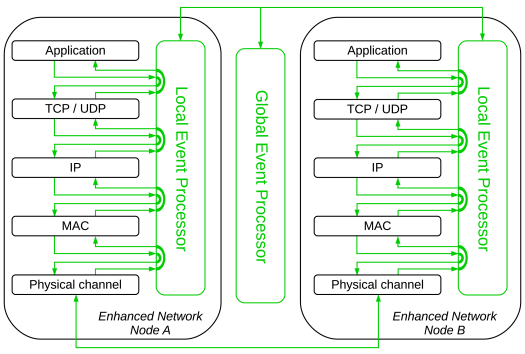


Fig. 3: Architecture of the *Attack Simulation Engine*.

The LEP module manages the attack events and operates transparently with respect to the other components of the *Enhanced Network Node* module. Specifically, the LEP module intercepts incoming and outgoing network packets traveling through a node's communication stack, acting as *gate-bypass* between each pair of INET modules implementing the different communication layers. Then, depending on the considered attacks to be evaluated, it can inspect and alter packets' content, inject new packets, or even discard intercepted ones. Finally, the LEP module can also alter the node's behavior at different layers, change its position in space, or even neutralize the node by making it inactive.

To address the presence of multiple network nodes and enable the simulation of complex attacks, we instantiate an *Enhanced Network Node* module for each network node, and a single *Global Event Processor* (GEP) module that connects all the *Enhanced Network Node* modules with one another. The GEP module is separately connected with every LEP module, so allowing them to synchronize and communicate with one another in order to implement more complex, possibly distributed, security attacks. Finally, the LEP and GEP modules handle packets at different communication layers and conveniently access their header fields by means of the OMNeT++ *descriptor* classes.

Figure 3 shows the overall architecture of the ASE, with reference to two interconnected network nodes. Our extensions integrate the *Local Event Processor* and *Global Event Processor* modules highlighted in green within the INET framework, in order to correctly manage simulation events and network packets. This requires particular attention for the SDN switches, to maintain the separation between the *Control plane* and the *Data plane*, and to correctly manage the multiple MAC interfaces (see Figure 2(b)).

Note that the ASE consists in additional components integrated within the INET framework to support the processing of attack events. That is, we do *not* fundamentally modify INET as to the handling and scheduling of simulation events, and we do *not* modify any of the available applications, communication protocols, or physical models. Most important, the user is *not* required to implement or customize any component of the simulation platform.

3) Injection of attack events: The attack description based on ASL is converted into a XML configuration file by means of a Python *Attack Specification Interpreter*, and then provided as input to INET upon simulation startup. Such configuration file is composed of three different sections, i.e a first part listing all the specified physical attacks, a second part listing all the specified conditional attacks, and a final third part listing all the specified unconditional attacks. At simulation startup, the ASE parses the XML configuration file and proceeds as follows. For each node n involved in at least one attack, the ASE:

- Creates one list LP_n, each element of which includes the description of one physical attack involving node n. The list elements are cronologically ordered according to the respective attack's occurrence time.
- Creates one list LC_n, each element of which includes the description of one conditional attack involving node n. The list elements are cronologically ordered according to the respective attack's starting time.
- Creates one list LU_n, each element of which includes the description of one unconditional attack involving node n. The list elements are cronologically ordered according to the respective attack's starting time.

After that, the ASE starts a number of timers, each one associated to a specified attack. That is, for each node n involved in at least one attack, the ASE:

- Creates a set of attack timers TP_n, each one of which associated to one physical attack involving node n.
- Creates a set of attack timers TC_n, each one of which associated to one conditional attack involving node n.
- Creates a set of attack timers TU_n, each one of which associated to one unconditional attack involving node n.
- Starts all the timers in TP_n, TC_n, and TU_n, in order to schedule the respective attack's occurrence.

Throughout the network simulation, the ASE proceeds as follows. When an attack timer associated to a node n expires, the ASE retrieves the associated attack A. Then:

- If A is a physical attack, the ASE executes the associated node primitive, and removes A from the attack list LP_n.
- If A is a conditional attack, from then on the ASE starts intercepting packets flowing through node n's communication stack, by means of node n's LPE. Intercepted packets are filtered, based on the condition specified in the conditional statement of attack A. For each packet that satisfies the conditional statement, the ASE executes the list of events described by the message primitives in A. The execution of some node primitives may involve also the GEP as well as the LEP modules of other nodes than n.
- If A is an unconditional attack, from then on the ASE starts executing the corresponding list of message primitives, and repeatedly performs A according to the occurrence frequency in the attack description. The GEP is responsible for starting the actual reproduction of uncondintional attacks.

IV. Evaluation of a Denial of Service attack

In this section, we simulate a simple Denial of Service attack against a server host, and present some preliminary results. We refer to the scenario in Figure 4, which includes: i) a SDN controller and a switch; and ii) four client hosts and three server hosts, running a UDP application. Besides, Client1 sends 10 packets per second to Server1; Client2 and Client3 send 5 and 3.33 packets per second to Server2, respectively; Client4 sends 5 packets per second to Server3.

Each flow installed on the switch expires every 30 seconds. When this happens, the switch notifies the SDN controller, removes the flow and possibly re-establishes it upon receiving new packets from/to the involved host(s). The SDN controller periodically collects flow statistics from the switch, according to a configurable interval I. Besides, the SDN controller runs a monitoring application that analyzes flow statistics in order to detect possible traffic anomalies. In particular, it relies on: i) entropy-based techniques for anomaly detection [11] [12]; and ii) bounded rates for transmission/reception of packets on a single-node basis. If traffic anomalies are detected on a given flow, the SDN controller sends an OpenFlow message OFPT_FLOW_MOD to the switch, in order to install a selective *drop* policy and discard all packets matching with that flow until further notice. In this scenario, we consider an adversary that has compromised Client3, and exploits it to transmit *additional* network packets to Server2, starting from time $t = 90$ s, and according to a packet injection rate R.

Figures 5 and 6 show the packet reception on Server2, considering the attack-free case "No attack" as baseline. A value plotted at second $t = s$ indicates the number of packets overall received by Server2 during the last second, i.e. between $t = s - 1$ and $t = s$. Specifically, Figure 5 considers

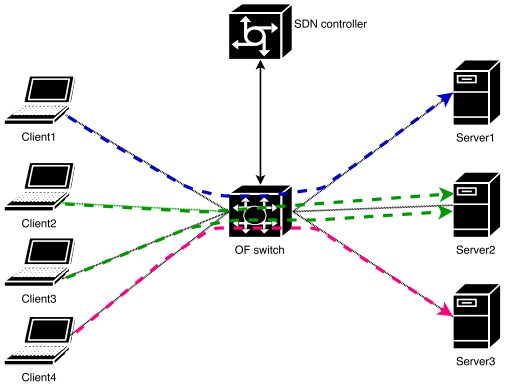


Fig. 4: Evaluated SDN network scenario.

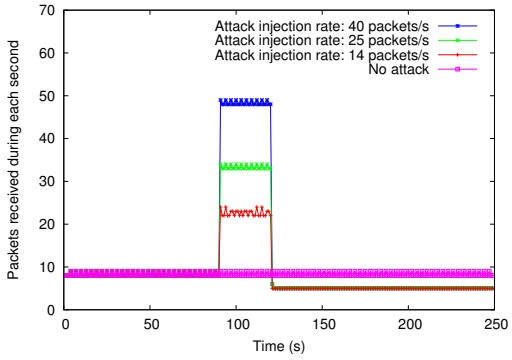


Fig. 5: Packet reception on Server2 (I=30 s).

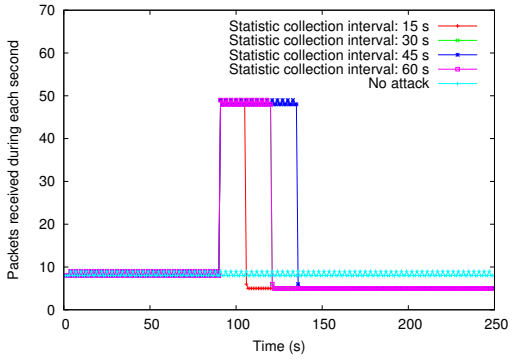


Fig. 6: Packet reception on Server2 (R=40 pkts/s).

different injection rates R, and shows that the greater R the more (attack) packets are received by Server2, until the traffic anomaly is mitigated at $t = 120$ s. From then on, the switch discards all packets coming from Client3 and addressed to Server2. Figure 6 considers different statistic collection intervals I, and shows the different times that it takes to detect and mitigate the attack, depending on the interval I considered by the SDN controller.

V. Conclusion

We have presented our extensions to the INET framework to support the evaluation of performance and security attacks in SDN scenarios. Our extensions allow the user to evaluate performance of a SDN architecture, assess accuracy and reactiveness of SDN-based monitoring systems, and quantitatively evaluate the impact of security attacks. We have evaluated a simple Denial of Service attack, and presented preliminary results. This work is an ongoing research activity, and we plan to propose it for an official contribution to the INET framework. Future work will focus on evaluating different classes of security attacks, considering different SDN-based monitoring systems and adversary models.

Acknowledgments

The authors would like to sincerely thank the anonymous reviewers and the shepherd Michael Kirsche for their insightful comments and suggestions that helped to considerably improve the technical quality of the paper. This project has received funding from the EIT Digital HII project ACTIVE, and the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no. 607109.

References

[1] B. Lantz, B. Heller and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *The Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, October 2010, pp. 1–6.

[2] D. Klein and M. Jarschel, "An OpenFlow extension for the OMNeT++ INET framework," in *6th International ICST Conference on Simulation Tools and Techniques (SimuTools '13)*, March 2013, pp. 322–329.

[3] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, January 2015.

[4] D. Pitt, "Open Networking Foundation," 2012. [Online]. Available: http://opennetworking.org

[5] INET Community. INET Framework Website. [Online]. Available: https://inet.omnetpp.org/

[6] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras V. Maglaris, "Combining OpenFlow and sFlow for an Effective and Scalable Anomaly Detection and Mitigation Mechanism on SDN Environments," *Computer Networks*, vol. 62, pp. 122–136, April 2014.

[7] M. Tiloca, A. Stagkopoulou, G. Dini, "INET_SDN_dev," 2016. [Online]. Available: https://github.com/marco-tiloca-sics/INET_SDN_dev

[8] M. Tiloca, F. Racciatti and G. Dini, "Simulative Evaluation of Security Attacks in Networked Critical Infrastructures," in *2nd International Workshop on Reliability and Security Aspects for Critical Infrastructure Protection (ReSA4CI 2015), published in Lecture Notes in Computer Science, LNCS 9338*. Springer International Publishing, September 2015, pp. 314–323.

[9] NS-3 Project. NS-3 v3.16 OpenFlow switch support. [Online]. Available: https://www.nsnam.org/docs/release/3.16/models/html/openflow-switch.html

[10] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," 2012.

[11] S. M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against SDN controllers," in *2015 International Conference on Computing, Networking and Communications (ICNC 2015)*, February 2015, pp. 77–81.

[12] S. Oshima and T. Nakashima and T. Sueyoshi, "Early DoS/DDoS Detection Method using Short-term Statistics," in *2010 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, February 2010, pp. 168–173.

[13] S.-Y. Wu and E. Yen, "Data Mining-based Intrusion Detectors," *Expert Systems with Applications*, vol. 36, no. 3, pp. 5605–5612, April 2009.

[14] T. Ahmed, B. Oreshkin and M. Coates, "Machine Learning Approaches to Network Anomaly Detection," in *Proceedings of the 2Nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, ser. SYSML'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–6.

[15] A. Vargas. OMNeT++ Community Website. [Online]. Available: http://www.omnetpp.org/

Towards a better battery model for INET

Laura Marie Feeney
Uppsala University
lmfeeney@it.uu.se

***Abstract*—Battery lifetime is often an important performance metric when simulating networks containing wireless devices. When evaluating system or protocol performance, there are cases where it is useful for the battery model to more accurately reflect the complex electrochemical processes involved in battery discharge.**

This paper describes an implementation of the KiBaM battery discharge model for INET. With regard to the INET implementation itself, the main technical issue is to extend the INET power model to more effectively handle both constant output voltage energy sources (mains power and simple battery models) and variable output voltage sources (complex battery models).

A custom battery testbed allows us to measure the discharge behavior of Li-coin cells under controlled conditions. The testbed is capable of coarse-grain modeling of the current draw associated with an IEEE 802.15.4 radio transmitting a frame. This enables a preliminary comparison between the simulation model and real discharge behavior.

I. Introduction

Operational lifetime is an important performance metric for battery-powered wireless devices, especially ones that are expected to operate for months or years without maintenance.

Such long lifetimes mean that experiments that directly measure battery lifetime under real loads are generally impractical. Instead, the device's power consumption is measured over a short period of time and it is this value that is used as the figure of merit. Usually, the operating system is instrumented to record the amount of time that device drivers spend in various states (e.g. sleep, transmit, receive). The power consumption is then inferred from hardware specifications for the device. There are also a few testbeds, such as [3], that directly measure the current that the device draws from the battery. However, this requires specialized hardware.

In both approaches, the focus is on the load that the device puts on the battery. The battery itself is treated as a linear source of charge (mA-h) that is drained by operations that consume i mA for t time.

In reality, a battery is an electrochemical system and a device fails when the chemical reactions in the battery are no longer able to maintain a sufficiently high output voltage under load (i.e. the device cut-off voltage). For a given battery chemistry and physical structure, the discharge process is determined by the timing and intensity of the applied load, as well as by external factors such as temperature. There is also considerable manufacturing variation, even between nominally identical batteries.

In short, battery behavior is complex and non-linear. This limits the value of "accelerated" testbed experiments that use unrealistically high loads and/or duty cycles to obtain a more practical run time: The battery discharge behavior is qualitatively different under these conditions.

This complexity also suggests that there are aspects of performance evaluation in which it may be important to consider battery discharge behavior in more detail. These include comparing protocol performance, absolute lifetime estimation, the accuracy of real-time state of charge estimation, and the battery/device interface. The last three of these are important practical topics which are particularly dependent on improved battery models. This work provides a step toward that goal.

Simulating battery lifetime requires modeling three components: the load created by the device, the battery's response to load, and the device's response to changes in battery state. This paper describes a preliminary implementation of the 'hybrid KiBaM' model [2] for OMNeT++'s [8] INET framework. The battery model is parameterized for the Panasonic CR 2032 Li coin cell [5], which is a 225 mA-h battery suitable for small devices.

The implementation of the load and device response are largely based on existing INET infrastructure for power modeling: The load is defined by tracking changes in device state, using OMNeT+'s signaling mechanism. The implementation and parameterization of the battery model itself are mostly independent of the OMNeT++/INET environment; the interested reader is referred to [7].

The main INET technical issue is to suitably extend the INET power model to more effectively handle both constant output voltage energy sources (mains power and simple battery models) and variable output voltage sources (complex battery models). Due to issues in the templates for managing units, it may be challenging to do this in a backwards compatible way.

A custom testbed for discharging batteries under controlled loads is described in [1]. The test hardware can be used to mimic the load patterns generated by an INET simulation and apply them to real batteries. The simulation results can then be compared to results obtained from the testbed.

II. Batteries and modeling

Battery modeling has been an important engineering technique in many fields for many years: A good overview for the non-expert is found in [6].

In most existing simulations of power consumption, the battery's residual capacity or state of charge is models using "coulomb counting". The battery has a known initial charge capacity (mA-h) which is drawn from the battery at a rate defined by the current $I(t)$. The consumed charge is the integral of current over time. In practice, $I(t)$ is treated as

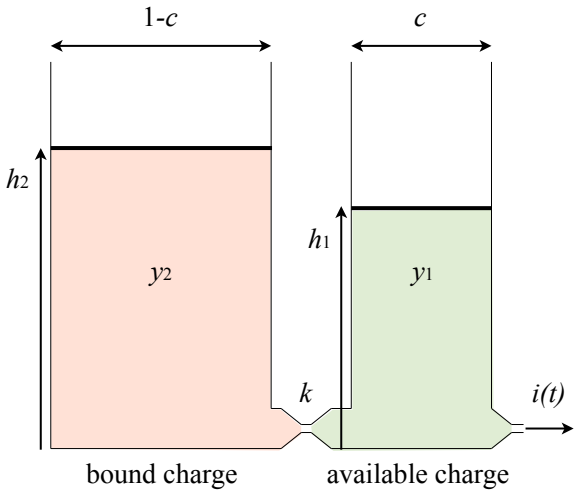


Fig. 1. Abstract view of the KiBaM model for battery state of charge.

a piecewise constant function that is the sum of fixed currents associated with each of the operations taking place on the device at any given time. The battery is empty and the device fails when the residual capacity reaches zero. (A power source, such as a solar panel, is similarly modeled as providing charge to the battery.)

The coulomb counting approach does not directly support modeling of the battery output voltage, which is mostly ignored. Generally, it is assumed to be equal to the battery's nominal output voltage as long as the battery has non-zero residual capacity and zero otherwise. In fact, it is the battery output voltage, and especially the voltage response when a load is placed on the battery, that actually determines operational lifetime of a device. The device electronics do not operate correctly when the input voltage is below some (device dependent) threshold value. Depending on the interface between the device and the battery, its behavior may also be affected by decreasing output voltage even before the cut-off.

A. KiBaM: State of charge modeling

Battery discharge exhibits two important non-linear behaviors[1]: The first is the rate-capacity effect: Discharging the battery at a higher current (rate at which charge is drawn from the battery) reduces its effective capacity. The second is charge recovery: An intermittent discharge is more efficient than a continuous one.

KiBaM [4] is an analytic battery model that is intended to reflect these behaviors by treating the battery as a pair of logical "wells", connected by a "valve" (Figure 1). One well contains "available" charge, which can used by the device. The current is the rate at which the charge is drained from this well. The battery fails when the device is unable to drain any more charge from the "available" well. The other well contains "bound" charge, which cannot be used by the device. Charge from the "bound" well flows into the "available" well though the "valve", at a rate proportional to the difference in height between the two wells. This interaction can be represented by a straightforward set of differential equations, which provide a good balance between accuracy and computational complexity.

Qualitatively, we see that this model captures several of the key points above: A high current discharges the battery disproportionately more quickly than a low current (the available well drains more quickly than it can be replenished by the bound well). An intermittent current discharges the battery more efficiently than a continuous one, since the available charge is replenished from the bound charge.

The KiBaM model is parameterized in terms of the relative sizes of the two wells and the valve between. The parameterization is highly battery specific and is based on detailed measurements of the battery in question under various kinds of controlled loads. The implementation reported here is parameterized for the Panasonic CR2032 Li coin cell.

B. Output voltage modeling

The ability of the device electronics to function as specified is determined by the battery's ability to maintain a sufficiently high output voltage under load. The battery output voltage can be seen as evolving on two different time scales. Under constant load, the output voltage is roughly constant or slightly decreasing over much of its lifetime, before decreasing sharply near its end of life.

Under an intermittent load, finer gain details emerge. When a load is applied to the battery, the output voltage immediately drops in proportion to its internal resistance, which increases as the battery is discharged. The output voltage continues to drop over the duration of the load; the magnitude of this effect reflects on the how well the electrochemical processes in the battery are able to "keep up" with the ongoing demand for charge. This ability decreases with decreasing state of charge, as the active species in the battery become depleted. It is the minimum voltage reached during this time that defines the battery lifetime. When the load is removed, the internal resistance of the battery is no longer present, so the output voltage increases. Due to relaxation and charge recovery processes in the battery, the voltage continues to recover for some time.

Hybrid KiBaM [2] combines the KiBaM state of charge model with an equivalent circuit model of the battery output voltage. The circuit consists variable voltage source (the battery's open circuit voltage), a resistor (the internal resistance) and an RC circuit. The circuit parameters are exponential functions of the KiBaM state of charge, with the constants determined experimentally for the battery in question.

Details of the Hybrid KiBaM model and its parameterization for the CR2032 Li coin cell are described in [7]. The results presented in that paper suggest the potential for significant improvements in the model. The voltage model is therefore only partially implemented in INET.

[1]Temperature dependence is another important factor, but is not addressed in this work.

III. INET MODEL

INET 3.2.4 includes a power consumption modeling framework, with key structures highlighted in Figure 2. The main components are the *EnergyStorage* and *Consumer* modules. A simple battery model is implemented in *SimpleEnergyStorage*.

The *EnergyStorage* module models the state of the energy store and makes this information available to other modules using OMNeT++ signals. It is where the battery discharge model itself is implemented. *EnergyStorage* can also invoke the *lifeCycleController* to propagate shut-down commands to simulate battery depletion. *Consumer* modules act as interfaces between *EnergyStorage* and other modules that simulate activities that consume power, such as a *Radio* module. There can be multiple consumers, which can be dynamically added and removed, *EnergyStorageBase* handles the relevant housekeeping[2].

Having a *Consumer* module separates modeling of the power consumption from the modeling of activities that consume power. This means that these modules do not need to explicitly include power consumption in their implementations, which allows for cleaner code. In particular, it allows the – possibly very hardware specific – calculation of the load caused by various operation to be developed separately. This means that each *Consumer* module is highly specific to the module whose power consumption it is representing. And of course, the module must somehow make the necessary information available to the *Consumer*. A *Consumer* can also control the level of detail with which it models the resulting load, allowing the *Consumer* to match its fidelity with that of the chosen *EnergyStorage*.

The *StateBasedConsumer* is part of the wireless physical layer and models the radio energy consumption. The *Radio* module emits signals announcing transitions among a range of radio states. The *StateBasedConsumer* queries the *Radio* module for detailed mode information and translates this into power consumption values. The power consumption of various radio modes is parameterized in the *StateBasedConsumer*.

The current implementation of the *StateBasedConsumer* assumes a fixed transmit power. To implement variable transmit power (and the associated differences in power consumption), it would be necessary for the *StateBasedConsumer* to also query the *Radio* for information about the transmit power being used when the radio is transmitting. The *StateBasedConsumer* would also need to be parameterized with relevant information about the power consumption of various transmit output powers.

The interface between *Consumer* and *EnergyStorage* that actually applies the simulated load to the simulated battery is instantiated via a function call *setPowerConsumption()*. This function takes Watts as its argument (as do related *get* and recording functions). This is deeply embedded in the power framework: the *EnergyStorageBase* classes in the INET power structure are defined in terms of Watts, via templates in the C++ code and @units in the .ned files.

The KiBaM battery model mathematics are straightforwardly implemented in a class derived from *Storage*. *Consumer* modules act as interfaces between *EnergyStorage* and modules that simulate activities that consume energy. However, KiBaM and other advanced battery models take current $I(t)$ as an input and generate either state-of-charge $SoC(t)$ and/or voltage $V(t)$ as an output.

The current KiBaM implementation deals with this incompatibility in rather unsatisfactory ways: *KiBaMEnergyStorage* provides a *set**Current**Consumption()* interface, which in turn requires a new current-based *KibamStateBasedConsumer*. To interact with the EnergyStorage "housekeeping" functions, the current implementation is based on some rather ugly code to bypass the Watts-oriented interfaces. (This cannot be used as a final implementation.)

Of course, it is possible to address this by implementing a parallel "current"-oriented class hierarchy. However, it would be helpful to be able to avoid duplicate code by creating a more integrated structure. The challenge is to make use of the safety and automatic conversions provided by the OMNeT++ @units templates, while still allowing for flexibility.

More generally, support for both power- and current-oriented power models is important for extending INET power modeling to other domains. In system and network simulation, there are (at least) two topic areas in which energy efficiency is important. The first area is reducing the power consumption of computing and communication infrastructure, such as data centers. These systems use mains power, which is intended to operate as a constant voltage energy source (e.g. 240V). In this case, consumption is naturally expressed in terms of Watts and Watt-hours. The second area is maximizing the effective lifetime of battery powered devices. A battery's output voltage depends heavily on its state-of-charge and load, as well as exogenous factors such as temperature. In this case, consumption is naturally expressed in terms of Amps and Amp-hours.

For simple battery models, this distinction is not an issue in practice because the battery voltage is treated as constant (the nominal output voltage) over its lifetime. Even if when loads are specified in terms of current, current is easily converted to power ($P = I \times V$) and this constant factor is simply carried through everywhere – the problem is only one of semantics.

However, more complex battery models expressly simulate the output voltage of the battery under load. Current is drawn from the battery at whatever voltage the battery supplies. This makes the use of power-based interfaces problematic. The relationship between the battery voltage and the current consumed for a given operation is non-trivial. In practice, it depends on device hardware and may be described in data sheets. This makes it important to have clean support for both battery modeling strategies, as well as to help ensure that users sensibly combine *EnergyStorage*, *Consumer*, and simulation modules.

[2] *EnergyStorage* is actually comprised of *EnergySource* and *EnergySink*, which handle power consumed from and added to the *EnergyStorage*, via *Consumer*s and *Generator*s, respectively. For non-rechargeable batteries, only the former is relevant and is the focus of this discussion. Codewise, the two interfaces are analogous.

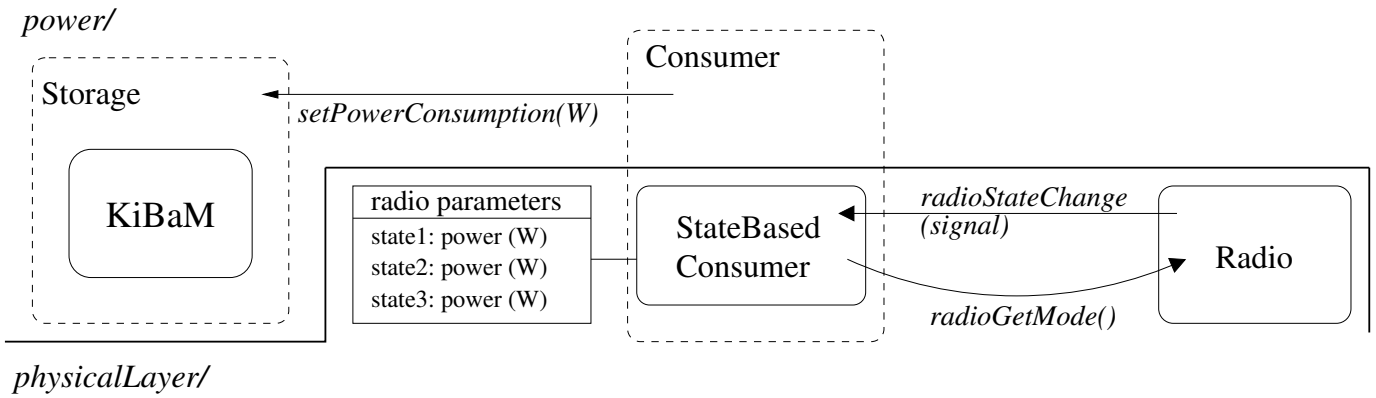


Fig. 2. Overview of the INET power framework. Power consumption is modeled by the *Consumer*, which translates information about device operations (e.g. of the *Radio*) into information about the load placed on the *Source*. Power generation is modeled using an analogous structure (not shown).

IV. Testbed

Because it is difficult to control the load generated by radios operating in real wireless environments, it is useful to be able to discharge batteries under more controlled conditions.

There are commercial testbeds that support current- and voltage- controlled loads and fine-grain measurements, but this approach is extremely expensive for large scale experiments. A large scale low-cost custom battery testbed is described in [1]. This hardware supports only resistive loads and allows only rather coarse grain load timing and voltage measurement.

In this work, the testbed is configured to generate load patterns that resemble those caused by transmissions from an IEEE 802.15.4 radio. To focus on the battery rather than variation associated with wireless communication, the load assumes that there is no contention or loss.

The load transitions through four operating states (Figure 3): First, it is in the receive state for one of eight randomly chosen intervals, ranging from 0.2 to 2.6 ms. This approximates the random backoff (n * 0.32 ms) and rx-to-tx transition. Then it is in the transmit state for 4.0 ms, the time it takes to transmit 125B. Then it returns to the receive state for 0.8 ms, which approximates the tx-to-rx transition and receiving the ACK. The nominal receive current is 20mA and the nominal transmit current is 28mA. Otherwise the load is 0mA, reflecting an idealized sleep mode. "Frames" are "transmitted" every 100ms, which results in an average duty cycle of about 7%.

The testbed was configured to discharge seven batteries over a period of 10 days. Figure 4 shows two sets of voltage measurements. The V_{oc} measurements are taken just before the load is applied and reflect the maximum recovery from the previous load. This is approximately the open circuit voltage. The V_{load} measurements are taken just before the load is removed and show the battery's decreasing ability to maintain output voltage under load.

The INET IEEE 802.15.4 model can be used to create a simulation scenario that puts a similar load on the simulated battery. To generate the simple load used in the testbed, the simulation consists of two *Host*s, a sender and a receiver,

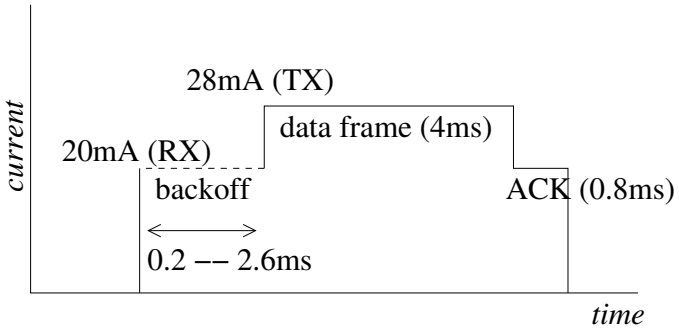


Fig. 3. Load used in simulation and in testbed experiments. Note that because the testbed provides a resistive load, rather than a constant current load, the actual current decreases over the lifetime of the experiment.

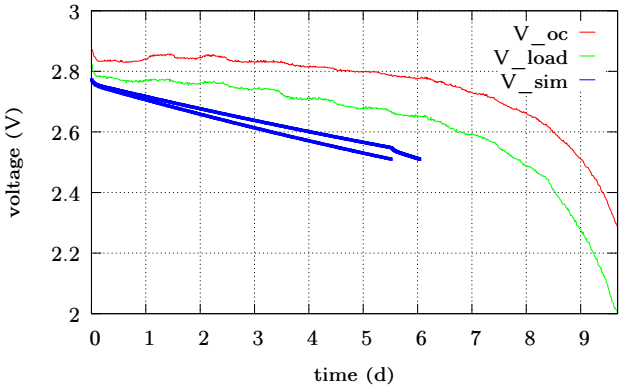


Fig. 4. Testbed and simulated voltages.

connected on an *idealChannel*. Every 100ms, the *Host*s' radios wake up and the sender's *udpAPP* module generates an 88B payload, to which IPv4 and IEEE 802.15.4 headers are added. The *ieee802154NarrowBandMAC* performs a random backoff, transmits (the transmit time is ∼4ms), then waits for and receives an ACK. The *Consumer* parameters are set to the same values that are used in the testbed, resulting in a similar battery load.

This simulation scenario is intended to exercise the battery model, rather than reflect the contention and collision behaviors, wakeup protocols, and multi-hop traffic that would be found in more realistic wireless network simulations and would result in more complex battery loads. In principle, it is possible to take any OMNeT++ simulation trace of the *Consumer* module and convert it to a scripted battery load for the testbed (modulo some testbed hardware limitations). (This is potential future work.)

Some substantial differences are visible between the two results. In particular, the KiBaM model takes the state-of-charge to zero before the experimental discharge and before the equivalent circuit model brings the open circuit voltage to null. The former discrepancy may be partly due to the non-current-controlled load in the testbed, which results in a lower effective load and longer lifetime. The latter discrepancy may be because, although the transient voltages are represented in the KiBaM model, they are not yet fully implemented in the equivalent circuit model. In other words, the battery has a measurable output voltage, but fails as soon as a load is applied – consistent with behavior observed in reality. Nevertheless, careful further study of the validity of the model and its parameterization for the Li coin cell are needed.

V. Conclusion

This paper has presented a preliminary implementation of the KiBaM hybrid battery model for INET. This is a step forward in improving INET's value as a tool for simulating and improving the behavior of networks of battery powered devices. There are, of course, still many limitations and open issues in the work.

With regard to the INET and simulation context, the most immediately visible issue is that the necessary changes to the INET power models is not backwards compatible – the current KiBaM implementation is an untenable hack. An alternative implementation that allows interfaces with different unit parameters would be most desirable.

To make meaningful use of a more detailed battery model also puts some requirements on simulation and model developers. The fundamental problem is ensuring that modeling of the load is appropriately detailed as well.

One factor is the impact of high currents: A current spike associated with a device state change might be ignored under a linear battery model because it consumes relatively few mA-h or occurs only infrequently. However, these operations might have a significant impact on the battery state. This can only be captured in the more sophisticated battery model if this aspect of the load is represented.

It also becomes more important to model all of the components of a device. When using simple battery models based on coulomb counting, the mA-h consumed by the radio, the sensor(s), and the processor could be treated independently. In particular, when evaluating or comparing the energy efficiency of various protocols, the latter could be ignored. A better understanding of both device loads and battery models is therefore needed to give appropriate guidance to developers and encourage more whole system simulation.

Independently of INET, there is considerable future work in improving and validating battery models, including extending the work to other types of batteries, especially rechargeable batteries. Another goal is incorporating temperature into models – this is particularly important for modeling SoC estimation and load balancing.

Acknowledgments

Christian Rohner has made many contributions to the analysis of the hybrid KiBaM model and the creation of the battery testbed. Parts of the OMNeT++ implementation were done in connection with the Computer Networking Project Course (Datakom III) at Uppsala University by students Felix Färjsjö, Andreas Gäwerth, Jonas Nilson, and Eric Stenberg.

References

[1] L. M. Feeney, C. Rohner, P. Gunningberg, A. Lindgren, and L. Andersson. How do the dynamics of battery discharge affect sensor lifetime? In *11th IEEE/IFIP Conf on Wireless On-demand Network Systems and Services (WONS)*, 2014.

[2] T. Kim and W. Qiao. A hybrid battery model capable of capturing dynamic circuit characteristics and nonlinear capacity effects. In *IEEE Transactions on Energy Conversion*, volume 26, 2011.

[3] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. Flocklab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *Information Processing in Sensor Networks (IPSN), 2013 ACM/IEEE International Conference on*, pages 153–165. IEEE, 2013.

[4] J. Manwell and J. McGowan. Lead acid battery storage model for hybrid energy systems. *Solar Energy*, 50(5), 1993.

[5] Panasonic. Manganese dioxide lithium coin batteries: Individual specifications. http://www.panasonic.com.

[6] R. Rao, S. Vrudhula, and D. Rakhmatov. Battery modeling for energy aware system design. *IEEE Computer*, 36(12), 2003.

[7] C. Rohner, L. M. Feeney, and P. Gunningberg. Evaluating battery models in wireless sensor networks. In *11th Int'l Conf on Wired/Wireless Internet Communications (WWIC)*, 2013.

[8] A. Varga. OMNeT++. In K. Wehrle, M. Günes, and J. Gross, editors, *Modeling and Tools for Network Simulation*. Springer-Verlag, 2010.

Automating large-scale simulation and data analysis with OMNeT++: lessons learned and future perspectives

Antonio Virdis, Carlo Vallati, Giovanni Nardini
Dipartimento di Ingegneria dell'Informazione, University of Pisa
Largo Lucio Lazzarino 1, I-56122, Pisa, Italy
a.virdis@iet.unipi.it, carlo.vallati@iet.unipi.it, g.nardini@ing.unipi.it

***Abstract*—Simulation is widely adopted in the study of modern computer networks. In this context, OMNeT++ provides a set of very effective tools that span from the definition of the network, to the automation of simulation execution and quick result representation. However, as network models become more and more complex to cope with the evolution of network systems, the amount of simulation factors, the number of simulated nodes and the size of results grow consequently, leading to simulations with larger scale. In this work, we perform a critical analysis of the tools provided by OMNeT++ in case of such large-scale simulations. We then propose a unified and flexible software architecture to support simulation automation.**

Keywords—OMNeT++; Large-Scale Simulations; Data Analysis; Simulation automation

I. INTRODUCTION

Nowadays simulation is a methodology widely used to drive the design and to assess performance of different computer systems. In computer networks in particular, simulation is widely adopted to drive the design of network or to assess the performance of existing deployments for provisioning or troubleshooting. Simulation models are exploited in place of real measurements or experiments for two main reasons: *(i)* simulation models can handle the complexity of such systems, characterized by many factors or settings that can influence the performance simultaneously; *(ii)* they can overcome the difficulty of studying systems that are distributed over distant areas and potentially all over the world.

In this context, OMNeT++ has gained popularity as a mature simulation tool. Especially in the area of networking, OMNeT++ is widely adopted by scientists and engineers that can exploit the availability of many simulation models for different network technologies, both wired and wireless.

Although simulation models are a simplified representation of actual systems, the increasing complexity of new communication technologies is currently pushing at a new different level the complexity of simulation models. Let us consider as example cellular networks: recent standards, e.g. LTE and LTE-Advanced, introduced new functionalities to handle the increasing demand for bandwidth and offer additional features to end users, with, however, a significant increase in complexity, which is necessarily reflected in the simulation models adopted, characterized by an overwhelming number of parameters, factors and number of simulated nodes.

Simulation models with a large number of factors and parameters usually imply simulation campaigns with a large number of different scenarios, aimed at evaluating the impact of each one on the overall system performance. Even though some techniques, e.g. factorial analysis [1], might be employed to reduce the number of scenarios, such simulation campaigns require a rigorous methodology to execute such large-scale experiments and, in particular, to analyze properly the large amount of results produced.

To this aim, software tools are usually employed to support the researcher to ensure a proper simulation workflow and eliminate - or minimize - biases or inaccuracies introduced by human operations, [2]. Specifically, tools that automate the execution of the simulation workflow and aid the researcher in the post-simulation analysis are usually employed. In this context, OMNeT++ already offers several tools and aids:

- An effective Graphical User Interface (GUI), which can be used to automate the execution of simulations. The end user can plan the simulation campaign through such interface exploiting an ad-hoc language adopted by OMNeT++ to configure the simulations and specify their parameters. The GUI can be used to run the experiments and monitor their progress through a graphical representation of network events.
- A post-simulation analysis GUI that can be exploited to visualize data and analyze metrics. Such GUI offers some basic data analysis operations, which can be exploited to produce simple graphs from simulation data.
- Some command line tools (opp_run) that drive and automate the execution of simulations without the GUI. Such tools can be used to run simulations on systems that lack of a graphical window system, e.g. a cluster or a server.
- A set of tools (scavetool) to export data from simulation results into different formats suited for external programs, e.g. Octave, Matlab, etc.

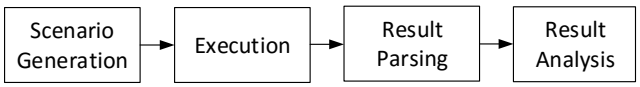


Figure 1 - Main operations to be performed during a large-scale experiment.

Although such functionalities are offered to automate the simulation workflow and aid researchers in post-simulation analysis, they have issues in handling large-scale simulation campaigns. Supporting complex simulations and analyzing large amount of results would require an improvement of the current software tools in order to ensure the rigorous execution of a consolidated simulation workflow. In this paper, we highlight the limits and issues of current tools included in the OMNeT++ suite in managing large-scale simulations and propose possible improvements in future perspective. Our goal is not to carry out a sterile analysis, but to trigger a fruitful discussion in the community on the best practices and their implementation, in order to improve future releases.

The remainder of the paper is organized as follows: in Section II simulation automation tools available for other network simulators are reviewed, in Section III the proper simulation workflow for large-scale simulations is presented along with an analysis of the automation tools currently available in OMNeT++. In Section IV we propose a software architecture for automating large-scale simulations, finally in Section V conclusions and future perspectives are presented.

II. Examples of Simulation Automation Tools

Network Simulator 2 (NS2) has been the standard de facto network simulator for years, before OMNeT++ and the release of its next major release NS3. Built around a simple basic architecture, NS2 became popular for the many network models made available over the years. Although popular, the simulator was completely lacking of a statistic collection framework and thus it offered no support for simulation automation or data analysis. To this aim, several third-party add-ons have been proposed over the year. Among them, it is worth to mention *ns2measure*, a framework designed to automate the collection of statistics, [3]. In addition, other extensions have been proposed to drive the simulation workflow and offer aid for post-simulation analysis. The *ANSWER* tool [2], for instance, is proposed to drive the experimental execution and automate the post-simulation data analysis through the aid of a GUI.

NS3, instead, provides native support for statistics collection and simulation flow management [7]. In particular, a set of internal modules are included to collect and store statistics, which can be saved on persistent storage (e.g. a database) or exported after simulation. Data provided by the *stat* module can be also exploited to check the status of simulations and drive their execution, e.g. stopping the simulation campaign when a certain level of confidence interval is reached. Although some functionalities for execution and data collection are offered, no support for post-simulation data analysis is provided natively as in OMNeT++.

III. Large Scale Simulation Workflow

Consider as an example of large-scale simulation a network with hundreds of nodes of different types. Each type has a set of metrics, in the order of tens, that are measured over time for each node. The network and its nodes can be configured with hundreds of parameters to tune their behavior. Our goal is to perform a simulation campaign to assess the performance of the network in various configurations. For this purpose, we define as *fixed parameters* all the parameters that will assume the same value during the whole simulation campaign; we define, instead, as *factors* those parameters whose value will be varied during the campaign and that will actually modify the system configuration.

The common workflow is described in Figure 1: first, the set of simulation scenarios required to include all (or a subset) of the possible combination of parameters is generated, then all the simulations are executed and results collected and parsed, finally results are analyzed. In the following, we overview more in details each single phase.

A. Scenario generation

The first step in the preparation of the simulation campaign is the generation of the scenario according to the parameters and factors. Within OMNeT++ this is done by means of *.ini* files, which can be modified in the IDE, either manually or through a form. Parameters are defined by simply assigning a value to them whereas factors are specified by means of the so called *iteration variables*, i.e. assigning an array of values to one single parameter. An example of the definition of one parameter and two factors is given in Figure 3. One of the most important factor is the number of *repetitions*, i.e. the number of times one configuration will be executed with different seeds for random number generation. Multiple repetitions are in fact used to perform independent replicas of the same scenario to increase the statistical soundness of the results, e.g., to improve confidence intervals.

Once all the above elements are defined, OMNeT++ will automatically translate the whole set of factors into a simulation campaign composed of N runs, one for each possible configuration, as we represent in the left part of Figure 2. If we indicate with $|fact_i|$ the number of values that are defined for factor-i, the total number of runs will be equal to:

$$N = \left(\prod_i |fact_i| \right) \times R$$

where R is the total number of repetitions. Each run is then associated with a unique numeric ID that will identify the single simulation all over the process. This approach leads to a run identification that is factor-agnostic, thus losing correlation between the run itself and the value of its factors. As we will see in the following, the latter is fundamental in the whole workflow as they define a run in a semantic way, representing how the system is configured. Note that the association between IDs and factors is preserved unless a factor is changed. It is quite common, for example, to change a factor during the

```
**.parameter = 50
**.factA = ${ 50 , 100 }
**.factB = ${ 1 , 2 }
```

Figure 3 - Parameters and factors definition in .ini files

ID	factA	factB
0	50	1
1	100	1
2	50	2
3	100	2

ID	factA	factB	repetition
0	50	1	0
1	50	1	1
2	100	1	0
3	100	1	1
4	50	2	0
5	50	2	1
6	100	2	0
7	100	2	1

Figure 2 - Example of mapping between run IDs and factors

campaign, e.g. adding one factor (or a value), or performing additional repetitions of the same configuration in order to reach the desired statistical confidence. Every time a factor is added or more repetitions executed, the whole set of IDs is modified accordingly, as shown in the right part of Figure 2. This, however, modifies the correspondence between IDs and factors, which might lead to errors when results are analyzed.

B. *Multiple run support*

Once the scenario is defined, the actual simulation campaign has to be performed, running the whole set of N runs. Modern computers are equipped with multi-core processors, which can be exploited for the execution of multiple parallel simulations. Two main options are available in this respect.

The OMNeT++ environment offers a tool for running multiple batch simulations within the same machine, the *opp_runall* tool. The latter can be executed either via IDE or via command line and can be configured in terms of various parameters, e.g. the set of runs to be executed and the number of parallel processes. However, the simulations to be run are specified through their IDs, and the corresponding set of factors has to be retrieved manually. Moreover, opp_runall can be used only to execute one configuration file at a time, thus parallel execution of multiple configurations has to be performed running two different instances.

The second tool that is available in this respect is *AKAROA* [4]. The latter is a powerful framework for parallel execution of multiple simulations in different computers. It also offers the possibility to monitor at run-time a set of metrics, and extend the duration until some defined criteria are met. Although extremely powerful, AKAROA has a non-negligible setup cost, as it needs to be integrated within the simulator code, e.g. modifying the statistic collection. A few works on the integration of AKAROA within OMNeT++ are available in literature, e.g. [5]. However, considering the most recent research works and the activity within the community, it does not seem to be actually used.

C. *Post simulation parsing*

After the whole campaign has successfully completed, results should be extracted and processed. One of the main problem with large-scale simulations is that they generate a considerable amount of result files, some of which can be very large. Parsing files can be cumbersome and also error prone.

The OMNeT++ environment has an extremely useful graphical tool for result extraction. First, it allows the selection of the set of files or folders to parse. Then, it has a powerful regular-expression based tool for parsing the results. The latter is extremely useful to quickly evaluate a small set of data. However, such tool does not scale with the size of results, i.e. it becomes extremely slow with large files and when the overall set of results becomes big. One common solution to the limits of the graphical interface in analyzing and extracting large volumes of simulation data is to exploit *scavetool*, a command line tool available to extract simulation data. Data, extracted in different format, can be imported in more powerful tools, e.g. Octave or Matlab, for analysis. However, *scavetool* has some limits, in particular when simulation scenarios with a large amount of data are considered, the tool is often unable to complete the extraction, as it requires all the data to be loaded in RAM. In addition, the extraction of a single metric or specific simulation scenarios is based on defining matching rules through regular expression, which is flexible but also error prone.

Another option available is to exploit R for data analysis. R is one of the most famous tool for statistical analysis. A plugin that allows to import directly in R simulation data is available. The researcher can first import data directly inside the R tool and then analyze the metrics and draw graphs. The main advantage of this approach is the large variety of statistical models and tools available in R, which makes possible the execution of any kind of analysis. The R tool, however, is not user-friendly and requires a non -negligible learning time. In addition, when very large simulation campaigns are considered, R cannot be used, as it requires all the data to be loaded in RAM.

When it comes to very large data sets, which cause memory issues to all the aforementioned solutions, the adoption of custom tools/scripts written by researchers is usually preferred, e.g. [8]. The realization of ad-hoc tools, however, is difficult, as simulation results are not stored in standard format, e.g. XML or JSON, and requires every time to re-invent the wheel.

D. *Results analysis*

In the previous section, we mentioned that OMNeT++ IDE provides an efficient tool for quick evaluation of simulation results. Although this is very useful during the testing phase of a new models or algorithms, it is not sufficient to show results in a graceful and statistically sound way, i.e. for adequate presentation in a research paper. For example, it lacks of the possibility to evaluate confidence intervals for the mean values. Moreover, several types of chart are not available in the OMNeT++ environment, e.g. box plots and cumulative distribution functions (CDFs) plots, which are widely used in

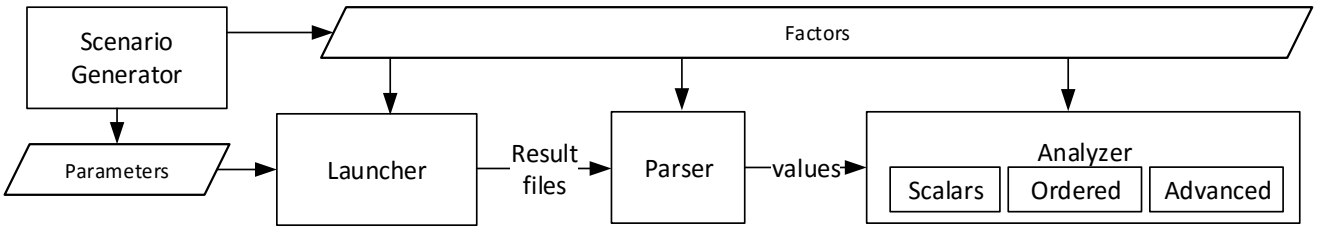


Figure 4 - Proposed architecture for large-scale experiment

the research community for representing distributions. In some cases, more advanced analysis must be performed, e.g. factorial analysis. Thus, results must be processed by external tools such as Gnuplot, etc. However, such programs require to build custom scripts that operate on results (or a subset of it) provided in a predefined format, which could lead to the same issues highlighted for the development of custom tools.

IV. A SOFTWARE ARCHITECURE FOR LARGE-SCALE SIMULATION

In this section, we will propose a software architecture for automating the execution of large-scale simulation campaigns. The purpose of this architecture is to serve as reference for future development, triggering a discussion within the community towards a consolidated point of view. We will take into account all the limitations highlighted in the previous sections and focus on four main goals:

1. define a modular and customizable structure;
2. use a factor-based indexing of configuration;
3. guarantee and improve statistical soundness of results;
4. ensure scalable performance.

We define four operational blocks, as represented in Figure 4, each one implementing one of the main steps of the simulation workflow described in Section III. The interactions between blocks occur using well-defined interfaces, but the internal structure of each of them can be customized.

Using Figure 4 as reference, the *scenario generator* creates two files: a first one containing all the fixed parameters of the campaign (the common .ini file generated through the OMNeT++ GUI); a second file, instead, containing all the factors with their values.

The *launcher* will take as input the parameters and the factors to execute the whole simulation campaign in parallel on a defined number of CPUs. The launcher should allow to execute selectively a subset of the simulation scenarios, e.g. for test or troubleshooting, with scenarios selected in a factor-based manner. This will allow one for example to execute all the campaigns where factor x has value y. The number of repetitions will be also configurable, allowing dynamic extension.

The output of the *launcher* is a set of result files, each one tagged with the values of all the factors, which will be made available to the *parser*. The latter will translate the output of the simulator in the format expected by the *analyzer*. The *parser* can have various implementations depending on the format of the result files. For example if standard .sca files are used, it can be implemented as a wrapper for scavetool, still maintaining the aforementioned scalability issues, but with limited development cost. More efficient solutions can be obtained creating custom parses for the .sca files (e.g. [8]) or through the definition of a new format for data files from scratch (e.g. binary files) or adopting a standard format (e.g. XML, JSON, etc.).

Regardless of its internal implementation, the parser will produce a set of results, tagged with the values of the factors. The *analyzer* in turn will use such files to perform three main operations:

1. compute scalar results such as mean values;
2. create ordered statistics which can be used to generate CDFs, scatterplots, etc.;
3. perform advanced statistical processing, such as factorial analysis.

The analyzer has access to the list of factors; thus it can be configured to selectively operate on a subset of the results. Its final goal is to produce results that are ready for representation, thus any tool can be used to create plots, such as Excel, Kaleidagraph and Calc, or batch ones such as Gnuplot.

V. CONCLUSIONS AND FUTURE PERSPECTIVES

In this work, we presented a critical analysis of the tools for simulation automation provided by the OMNeT++ framework. We focused on the context of large-scale simulations and discussed the limitation of such tool in each step of the simulation workflow. Finally, we proposed a software architecture for large-scale simulation, with the aim of serving as guideline for future development within the community. Our main goal is to trigger a discussion with all the members of the OMNeT++ community, and share our view on the subject.

REFERENCES

[1] C. Cicconetti, E. Mingozzi, C.Vallati, "A 2k · r factorial analysis tool for ns2measure", In Proceedings of VALUETOOLS 2009.

[2] M. M. Andreozzi, G. Stea, C. Vallati, "A framework for large-scale simulations and output result analysis with ns-2", Simutools 2009.

[3] C. Cicconetti, E. Mingozzi, G. Stea. "An integrated framework for enabling effective data collection and statistical analysis with ns-2", WNS2 2006.

[4] K. Pawlikowski, V. W. C. Yau and D. McNickle, "Distributed stochastic discrete-event simulation in parallel time streams," Simulation Conference Proceedings, Winter, Lake Buena Vista, FL, USA, 1994.

[5] S. Sroka, H. Karl, "Using Akaroa2 with OMNeT++", 2nd OMNeT++ Workshop, Berlin, Germany, Jan 2002.

[6] http://www.r-project.org

[7] https://www.nsnam.org/docs/manual/html/statistics.html

[8] https://github.com/sommer/inet-sommer/tree/analysis/etc

WiFi-Direct Simulation for INET in OMNeT++

Syphax Iskounen, Thi-Mai-Trang Nguyen and Sébastien Monnet
Sorbonne Universités, UPMC Université Paris 6, CNRS, UMR 7606, LIP6
4 Place Jussieu, 75005 Paris, France
{Syphax.Iskounen, Thi-Mai-Trang.Nguyen, Sebastien.Monnet}@lip6.fr

***Abstract*—Wi-Fi Direct is a popular wireless technology which is integrated in most of today's smartphones and tablets. This technology allows a set of devices to dynamically negotiate and select a group owner which plays the role access point. This important feature is the strength of Wi-Fi Direct and makes it more and more widely used in telecommunications networks. In this paper, we present the implementation of Wi-Fi Direct in the INET framework of OMNeT++. We have implemented the main procedures of Wi-Fi Direct such as discovery, negotiation and group formation. The implementation has been validated by two test scenarios which show the conformity of the implementation to the protocol specification.**

Keywords—Wi-Fi Direct; OMNeT++; network simulation.

I. INTRODUCTION

With the emergence of mobile computing and Internet of Things, new applications and services have been developed for smartphones and tablets requiring an easy way to quickly create network connection between devices. These applications usually consume more bandwidth than the traditional voice services and have been designed for data transmission between nearby devices. The concept of Device-to-Device (D2D) communication has been introduced as a means to offload the cellular macro cells and facilitate direct communications between nearby devices [1].

In the Wi-Fi technology, there were historically two communication modes in the 802.11 standard, the infrastructure mode and the adhoc mode. The infrastructure mode based on the deployment of Access Points (AP) is currently used everywhere - in a campus, offices, at home or in public hotspots - to provide Wi-Fi users with ubiquitous Internet connection. The adhoc mode allows two Wi-Fi devices to directly communicate without the necessity of having an access point. This mode is used widely as the link layer for mobile adhoc networks in conjunction with an adhoc network routing protocol in order to provide a mutihop communication. However, for several reasons such as complexity of use and lack of power saving, the adhoc mode never becomes widely deployed nor used in practical wireless networks [2]. To address the need of easily and quickly setting up D2D data communications, the Wi-Fi Alliance has developed the Wi-Fi Direct technology [3]. Today, Wi-Fi Direct is not only the de facto technology for direct communications between Wi-Fi devices but also a strong candidate for D2D communications in future 5G networks.

Wi-Fi Direct was inspired from the infrastructure mode. The devices form groups. In each group, one selected node assumes the AP functionality. The advantage of Wi-Fi Direct is the inheritance of the enhanced QoS, power saving and security mechanisms from the infrastructure mode. In addition, a large number of applications designed for connecting devices such as file sharing are available for mobile users and the APIs for Android are available for developers [4].

The research community has shown a big interest in Wi-Fi Direct. In home networking, Wi-Fi Direct enabled devices can collaborate and share streaming-based media content [5]. In vehicular networks, Android-based smartphones can use Wi-Fi Direct for fast and cheap inter-cars communications [6]. In mobile social networks, users in proximity can automatically discover each other by Wi-Fi Direct and enable a variety of interactions such as chat or games [7]. In cellular networks, the operator network can assist neighboring client devices to activate Wi-Fi Direct and communicate via the D2D links instead of cellular links in order to offload the macro cell and enhance the performance of cellular communications [1, 8].

As Wi-Fi Direct is integrated in most handheld devices, most research contributions are validated by prototyping. However, we argue that it is important for network simulators to support Wi-Fi Direct allowing researchers to evaluate the performances of the network in complex scenarios with a large number of nodes and design protocols before prototyping. In this paper, we present our implementation of Wi-Fi Direct in the INET framework version 3.2.1 of OMNeT++ version 4.6 [9], an open-source network simulator widely used in academia. The implementation consists of a new module simulating the operation of Wi-Fi Direct.

The remainder of the paper is organized as follows. In section II, we present the architecture and functionalities of Wi-Fi Direct as well as the current implementation of the IEEE 802.11 in OMNeT++. In section III, the details of Wi-Fi Direct implementation are provided. We test the implementation in section IV. Finally, we conclude the paper in section V.

II. BACKGROUND

A. *Wi-Fi Direct Architecture*

Wi-Fi Direct is also called Wi-Fi Peer-to-Peer (P2P) [3]. A P2P group is a set of Wi-Fi Direct devices consisting of one P2P Group Owner (GO) and zero or more Clients as illustrated in Fig. 1.

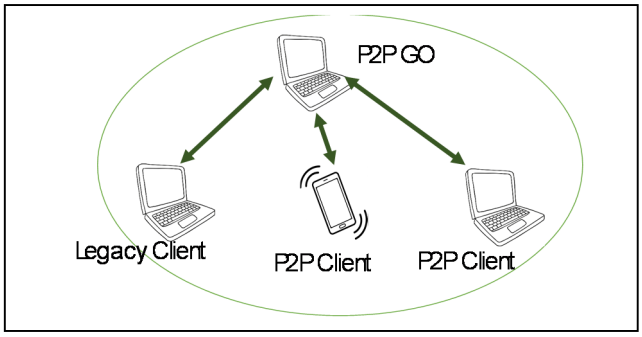


Fig. 1. Example of a Wi-Fi Direct Group.

The GO plays the role of an access point to provide clients with connectivity. It is also a communication node and can be a source or a destination of higher layer applications. All Wi-Fi Direct devices implement both roles of client and AP. These roles are dynamically assigned during the group formation. After the group formation, the P2P GO works as an AP and announces its presence by periodically sending Beacon frames. P2P devices receiving the Beacon frames can connect to the group as in a traditional Wi-Fi network. Devices that are not P2P compliant can also join the group as Legacy Clients (Fig. 1).

A P2P Group can be formed in three ways: standard, autonomous and persistent [2]. The *standard* case as shown in Fig. 2 consists of two phases, discovery and group formation.

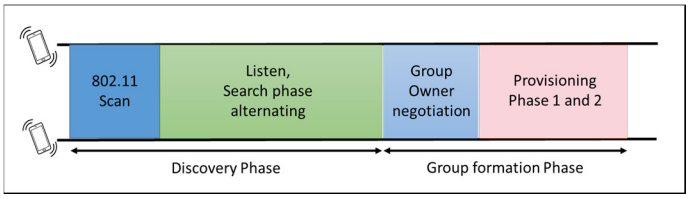


Fig. 2. Standard group formation

In the *discovery* phase, a device starts by a normal *802.11 scan* to discover existent P2P groups and traditional Wi-Fi APs. If there is no already established group, the device alternates between the *search* and *listen* states to discover another Wi-Fi Direct device which also wishes to form a group. In the search state, the device performs an active scan by sending *Probe Requests* in each channel. In the *listen* state, the device listens on a channel for a *Probe Request* to respond by a *Probe Response*. Once the two P2P devices have discovered each other by sending *Probe Request* and receiving *Probe Response* over the same channel, they can move on the group formation phase.

In the *group formation* phase, the two devices start by the *Group Owner negotiation* to know which one will become GO using a three-way handshake as illustrated in Fig. 3. It is worth noting that a Wi-Fi Direct Group may contain more than two devices but only the two first devices can discover each other and negotiate for a GO. The other devices simply detect the presence of GO and joins an existing group. In the *GO Negotiation Request* and *Response* messages, there is a *GO Intent* value declared by each participating device. The device declaring the highest value becomes the P2P GO. Once the role of GO has been agreed, the two devices switch to the *Provisioning phases 1 & 2* to establish a secure communication. In the *Provisioning phase 1*, security keys for mutual authentication are created [10]. In the *Provisioning phase 2*, the device playing the role of client reconnects to the GO using the new authentication credentials.

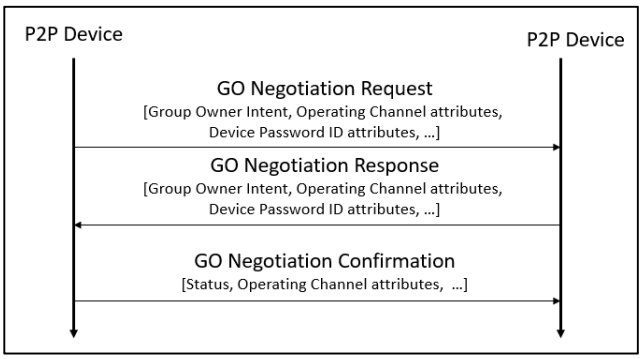


Fig. 3. Group Owner Negotiation message exchange [3]

A P2P device may create a P2P Group for itself and become immediately P2P GO by selecting a channel and sending Beacon frames. This way to form a P2P Group is called *autonomous*. Other devices can discover this Group by performing 802.11 scan and move directly on the Provisioning Phases 1 & 2 for establishing a secure communication within the group.

A group can be declared as *persistent* during first formation. A flag sent in Beacon, Probe and GO negotiation messages can perform this declaration. The devices of a persistent group store the network credentials and the role negotiated. Later on, when a device has leaved the group and wants to discover or establish a group, it can easily recognize that it has established a persistent group with the peer in the past. After the Discovery phase, the device can directly go to the Provisioning phase 2 using the previous network credentials. This way to form a P2P Group is called *persistent*.

B. 802.11 implementation in OMNeT++

The IEEE 802.11 infrastructure and adhoc modes have been implemented in the INET framework of OMNeT++ [11]. An IEEE 802.11 interface has four layers: agent, management, Medium Access Control (MAC) and physical layer (radio) as shown in Fig. 4.

The *physical layer* (the `Ieee80211Radio` module) models the radio propagation characteristics of the medium. The *MAC* layer (the `Ieee80211Mac` module) implements the Carrier Sense Multiple Access and Congestion Avoidance (CSMA/CA) protocol. The *management* layer implements management functions and generates management frames such as *Beacon*, *Probe*, *Authentication* and *Association* frames. This layer has several modules (e.g. `Ieee80211MgmtAdhoc`, `Ieee80211MgmtAP`, `Ieee80211MgmtSTA`) according to

the role of the device (station, AP or adhoc node). The agent layer (the `Ieee80211AgentSTA` module) is currently only present for 802.11 stations and allows the user to control the behavior of a station in the network. Researchers can implement a handover strategy using this module for example.

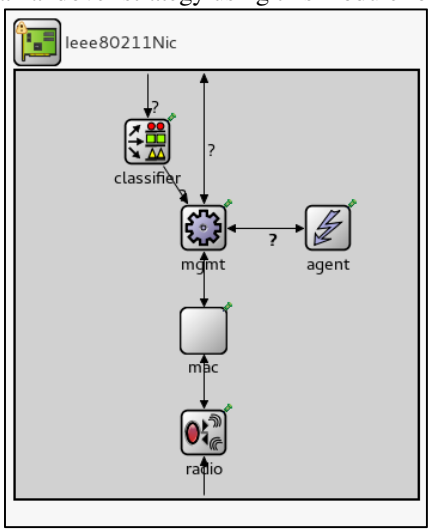


Fig. 4. IEEE 802.11 network interface module in INET Framework

III. Wi-Fi Direct Implementation

In order to implement the functionalities of Wi-Fi Direct, we have created a new module at the *management* layer called `Ieee80211MgmtSTAWifiDirect` [12]. Our model is compatible with INET Framework version 2.0.0 which requires OMNeT++ version 4.2 or later because the 802.11 classes that we have modified are present from this version of the INET Framework. In fact, Wi-Fi Direct corresponds to a new communication mode of Wi-Fi networks beside the existent infrastructure and adhoc modes. An 802.11 interface plays a new role, the Wi-Fi Direct peer, which is different from the existent roles of station, AP and adhoc node. A Wi-Fi Direct peer is able to scan, discover the peer, negotiate the group owner and establish a secure communication within the group as presented in section II.

The `Ieee80211MgmtSTAWifiDirect` module performs the encapsulation and decapsulation of data frames. It exchanges management frames to perform the group formation and group joining procedures as described in Fig. 5 and Fig. 6 respectively.

Fig. 5 presents the detailed message exchange for a standard group formation. The *scan* phase corresponds to the regular IEEE 802.11 scan procedure already implemented in the `Ieee80211MgmtSTA` module. After sending Probe requests without detecting any existent GO, the Wi-Fi Direct devices alternates between the *listen* and *search* states in the Find phase until they discover each other by sending *Probe Requests* and receiving *Probe Responses* over the same channel (channel 6 in this example). Three new frames, *GO Negotiation Request/Response/Confirmation* have been implemented conforming to the protocol specification to perform the *GO negotiation* phase. For the sake of simplicity and because our internal research objective is not really related to the security aspect, we have only implemented representatively the *Provisioning phases 1 & 2* by exchanging a number of N (e.g. N = 20) authentication frames. The detailed implementation of the authentication and key exchanges in the *Provisioning* phase can be subject to a future contribution in the OMNeT++ community.

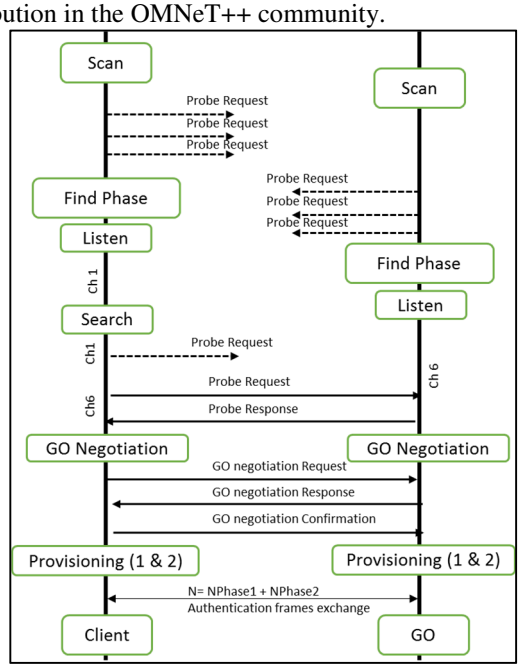


Fig. 5. Message exchange for standard group formation

Fig. 6 presents the detailed message exchange between a Wi-Fi Direct device detecting a group and the Group owner. After the scan phase (an active scan in this example), the new P2P device detects the GO of a group already formed. The P2P device asks to join the group by sending a *Provision Discovery Request*. Upon the reception of the response, it moves to the Provisioning phases 1 & 2.

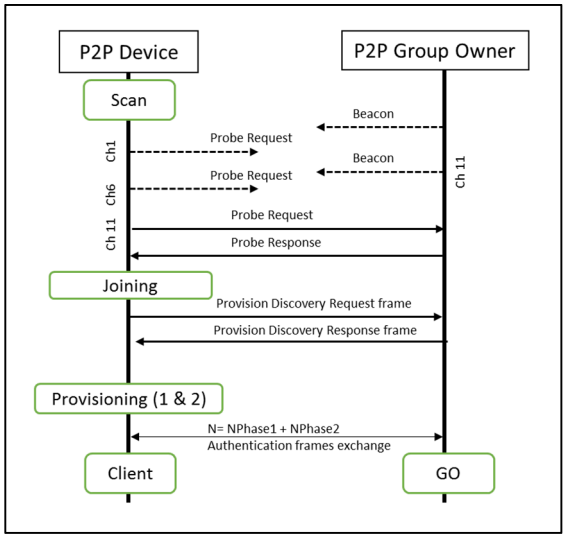


Fig. 6. Message exchange for joining an existent group

IV. Evaluation

In this section, we evaluate the implementation of Wi-Fi Direct by two test scenarios. In the first scenario, we create three Wi-Fi Direct hosts and let them automatically form the P2P Group following the standard group formation procedure. In the second scenario, we also create three Wi-Fi Direct host but one of them is assigned as a GO. That means this node will form a group autonomously for itself at the beginning of the simulation. As a consequence, two other hosts detect an existing GO and join the group. Sending Ping messages tests the connectivity within a group.

A. *Test of standard group formation*

Fig. 7 shows the topology of the first test scenario with 3 Wi-Fi Direct hosts created.

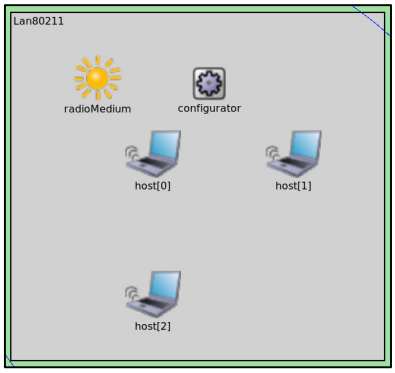


Fig. 7. Topology of Wi-Fi Direct network

In this topology we have a radio medium, an IPv4 Configurator for IP address assignment and three hosts of type `WirelessHost` with their manager module set to `Ieee80211MgmtSTAWifiDirect`. The `wirelessHost` used can be found in the INET project at `inet.node.inet.WirelessHost`.

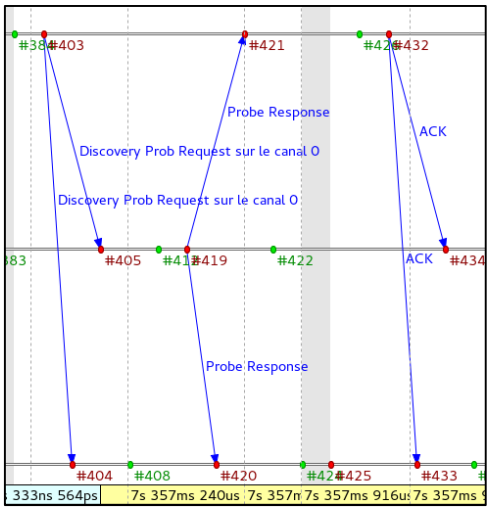


Fig. 8. Discovery phase

In the Discovery phase, each host firstly scans and doesn't detect any GO available. Then, they switch to alternating between the listen and search states to try to discover another Wi-Fi Direct host. Fig. 8 presents partially the *Discovery* phase when host[0] and host[1] discover each other. We can see that host[0] sends a *Discovery Probe Request* in channel 0 at event #403. Host[1] is in the listen state over this channel at that time and responds with a *Probe Response* message at event #419. Host[0] sends an acknowledgement (ACK) frame at event #432 and moves on the *GO negotiation* phase.

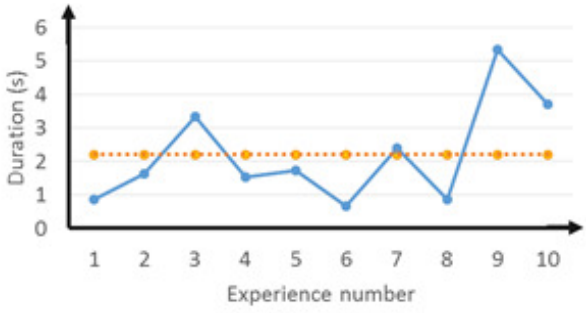


Fig. 9. Duration of the discovery phase

Fig. 9 shows the duration of the discovery phase that we have measured by our simulations. The average discovery time is about 2-3 seconds. This result corresponds the average discovery time that we have experienced with two smartphones supporting Wi-Fi Direct.

#288	6.40348094346	host[1] --> host[0]	GO Negotiation Request Frame
#288	6.40348094346	host[1] --> host[2]	GO Negotiation Request Frame
#301	6.40379541519	host[0] --> host[1]	ACK
#301	6.40379541519	host[0] --> host[2]	ACK
#316	6.40414941519	host[0] --> host[1]	GO Negotiation Response Frame
#316	6.40414941519	host[0] --> host[2]	GO Negotiation Response Frame
#329	6.40446388692	host[1] --> host[0]	ACK
#329	6.40446388692	host[1] --> host[2]	ACK
#374	6.405726554048	host[1] --> host[0]	GO Negotiation Confirmation Frame
#374	6.405726554048	host[1] --> host[2]	GO Negotiation Confirmation Frame
#386	6.406041025778	host[0] --> host[1]	ACK
#386	6.406041025778	host[0] --> host[2]	ACK
#414	6.431719516371	host[1] --> host[0]	Beacon
#414	6.431719516371	host[1] --> host[2]	Beacon

Fig. 10. The GO negotiation phase

Fig. 10 presents the GO negotiation phase with the three-way handshake between host[0] and host[1]. At the end, host[1] becomes the GO and starts sending *Beacon*s. Host[0] finishes the group formation by completing the *Provisioning phases 1 & 2* with host[1]. We have implemented these phases by the exchange of a number of *Authentication* frames which can be configured in the simulation configuration `.ini` file. Upon the reception of Beacon frames sent by host[1], host[2] starts the joining procedure to join the group.

Fig. 11 shows the test of connectivity within the group by sending a Ping request from host[0] to host[2] (event #2362) and another Ping request from host[1] to host[0] (event #2493). For the first Ping request, we can see that host[1] plays the role of AP relaying the Ping request from host[0] to host[2] (event #2390). The Ping reply is also transferred to host[0] via host[1], the GO of the group (event #2450). For the

second Ping request, host[0] responds by a Ping reply directly (event #2525).

#2342	9.531719516371	host[1] --> host[2]	Beacon
#2362	9.548813502304	host[0] --> host[1]	ping9
#2362	9.548813502304	host[0] --> host[2]	ping9
#2375	9.549495974034	host[1] --> host[0]	ACK
#2375	9.549495974034	host[1] --> host[2]	ACK
#2390	9.549949974034	host[1] --> host[0]	ping9
#2390	9.549949974034	host[1] --> host[2]	ping9
#2407	9.550632307598	host[2] --> host[0]	ACK
#2407	9.550632307598	host[2] --> host[1]	ACK
#2422	9.551066307598	host[2] --> host[0]	ping9-reply
#2422	9.551066307598	host[2] --> host[1]	ping9-reply
#2435	9.551748641162	host[1] --> host[0]	ACK
#2435	9.551748641162	host[1] --> host[2]	ACK
#2450	9.552122641162	host[1] --> host[0]	ping9-reply
#2450	9.552122641162	host[1] --> host[2]	ping9-reply
#2465	9.552805112892	host[0] --> host[1]	ACK
#2465	9.552805112892	host[0] --> host[2]	ACK
#2493	9.592844616388	host[1] --> host[0]	ping9
#2493	9.592844616388	host[1] --> host[2]	ping9
#2510	9.593527088118	host[0] --> host[1]	ACK
#2510	9.593527088118	host[0] --> host[2]	ACK
#2525	9.593961088118	host[0] --> host[1]	ping9-reply
#2525	9.593961088118	host[0] --> host[2]	ping9-reply
#2540	9.594643559848	host[1] --> host[0]	ACK
#2540	9.594643559848	host[1] --> host[2]	ACK
#2561	9.631719516371	host[1] --> host[0]	Beacon

Fig. 11. Testing the connectivity for the first test scenario

B. *Test of autonomous group formation*

In the second test scenario, we use the same topology as in the first test scenario: three Wi-Fi Direct hosts, a radio medium and an IPv4 Configurator as presented in Fig. 7. The `.ini` file, presented in Fig. 12, allows us to configure host 0 as a GO so that it will perform an autonomous group formation at the beginning of the simulation with the SSID `''Wi-Fi Direct Group''`.

```
# ping app host[0] pinged by Host[1]
**.numPingApps = 1
*.host[1].pingApp[0].destAddr = "host[0]"
*.host[1].pingApp[0].sendInterval = 1s
# ping app host[1] pinged by host[2]
*.host[2].pingApp[0].destAddr = "host[1]"
*.host[2].pingApp[0].sendInterval = 1s
#Configure the P2P Group
**.host[0].wlan[0].mgmt.WiFiDirectUsed=true
**.host[0].wlan[0].mgmt.WiFiDirectGO=true
**.host[0].wlan[0].mgmt.strGroup="Groupe Wifi Direct"

**.host[1].wlan[0].mgmt.WiFiDirectUsed=true
**.host[1].wlan[0].mgmt.WiFiDirectGO=false
**.host[1].wlan[0].mgmt.strGroup="Groupe Wifi Direct"

**.host[2].wlan[0].mgmt.WiFiDirectUsed=true
**.host[2].wlan[0].mgmt.WiFiDirectGO=false
**.host[2].wlan[0].mgmt.strGroup="Groupe Wifi Direct"
```

Fig. 12. Configure host 0 as GO using .ini file

We also allow the association of a host to a specific group created by an autonomous GO via the .ini file. This feature has been implemented for our internal research purpose in which we can try different group formation strategies. For example, the Wi-Fi Direct groups can be formed in a guided manner with an advanced power control technique to reduce the interference level and improve the network performances. As illustrated in Fig. 12, we configured host[1] and host[2] to be in the same group as the group created by host[0]. In practice and in other simulation scenarios, researchers can let other hosts to randomly select the group to join if there are several groups available.

#80	5.084426574409	host[0] --> host[1]	Beacon
#80	5.084426574409	host[0] --> host[2]	Beacon
#121	5.085398907973	host[2] --> host[0]	Provision Request
#121	5.085398907973	host[2] --> host[1]	Provision Request
#134	5.085713241537	host[0] --> host[1]	ACK
#134	5.085713241537	host[0] --> host[2]	ACK
#149	5.086107713267	host[1] --> host[0]	Provision Request
#149	5.086107713267	host[1] --> host[2]	Provision Request
#162	5.086422184997	host[0] --> host[1]	ACK
#162	5.086422184997	host[0] --> host[2]	ACK
#177	5.086796184997	host[0] --> host[1]	Provision discovery Response
#177	5.086796184997	host[0] --> host[2]	Provision discovery Response
#190	5.087110518561	host[2] --> host[0]	ACK
#190	5.087110518561	host[2] --> host[1]	ACK
#205	5.087504852125	host[0] --> host[1]	Provision discovery Response
#205	5.087504852125	host[0] --> host[2]	Provision discovery Response
#218	5.087819323855	host[1] --> host[0]	ACK
#218	5.087819323855	host[1] --> host[2]	ACK

Fig. 13. Autonomous group formation and joining procedure

As presented in Fig. 13, host[0] starts to send *Beacon* frames when we launch the simulation to announce its presence. The other hosts, at the reception of the *Beacon* frames sent by host[0], start the joining procedure by sending the *Provision Discovery Request* frame. The P2P GO (host[0]) replies with the *Provision Discovery Response* frame. After the joining procedure, the *Provisioning phases 1 & 2* happen (that we do not show here) as usual to accomplish the group formation.

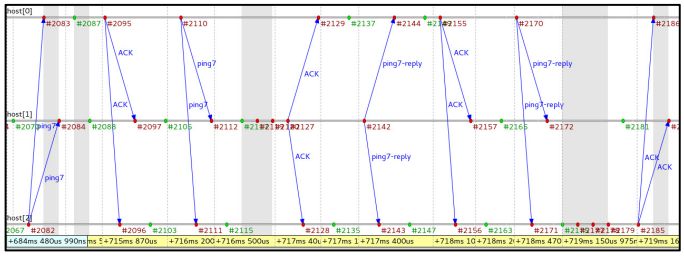


Fig. 14. Connectivity test for scenario 2

Fig. 14 shows the connectivity test for the group. Host[2] sends a Ping request to host[1] via host[0], the GO. Similarly, the Ping reply sent from host[1] to host[2] is also relayed by the GO.

V. Conclusion

In this work, we have implemented Wi-Fi Direct in the INET Framework of OMNeT++ [12]. The functionalities of Wi-Fi Direct are implemented as a management module in the same way as other IEEE 802.11 management modules already existing in the INET Framework in order to facilitate its utilization, its customization and its integration in OMNeT++. This implementation can be used for research on networking using Wi-Fi Direct such as protocol design and performance evaluations. In future works, we will integrate our modules into the INET Framework and use them for the research on Wi-Fi Direct-based device-to-device communications in dense wireless networks to offload the Wi-Fi network infrastructure.

References

[1] A. Asadi and V. Mancuso, "WiFi Direct and LTE D2D in action", IFIP Wireless Days (WD), Valencia, Spain, November 2013.

[2] D. Camps-Mur, A. Garcia-Saavedra and P. Serrano, « Device-to-Device Communications with WiFi Direct:

Overview and Experimentation », IEEE Wireless Communications, Vol. 20, No. 3, June 2013.
[3] Wi-Fi Alliance, "Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.5", August 2014.
[4] Wi-Fi Alliance, URL: http://www.wi-fi.org
[5] H. Yoon and J. Kim, "Collaborative Streaming-based Media Content Sharing in WiFi-enabled Home Networks", IEEE Transactions on Consumer Electronics, Vol. 56, No. 4, November 2010.
[6] A. Djajadi and R.J. Putra, "Inter-cars Safety Communication System Based on Android Smartphone", IEEE Conference on Open Systems (ICOS), Subang, Malaysia, October 2014.
[7] J. Zuo, Y. Wang, Q. Jin and J. Ma, "HYChat: A Hybrid Interactive Chat System for Mobile Social Networking in Proximity", IEEE International Conference on Smart City (SmartCity), Chengdu, China, December 2015.
[8] A. Pyattaev, K. Johnsson, A. Surak, R. Florea, S. Andreev and Y. Koucheryavy, "Network-assisted D2D Communications: Implementing a Technology Prototype for Cellular Traffic Offloading", IEEE Conference on Wireless Communications and Networking (WCNC), Istanbul, Turkey, April 2014.
[9] INET Framework, URL: https://inet.omnetpp.org/
[10] Wi-Fi Alliance, « Wi-Fi Simple Configuration Technical Specification, Version 2.0.5 », August 2014.
[11] OMNeT++, « INET Framework Manual – Chapter 9: The 802.11 Model", January 2016.
[12] Wi-Fi Direct Implementation Source Code. URL: http://www-phare.lip6.fr/~trnguyen/research/wifidirect

Simulating device-to-device communications in OMNeT++ with SimuLTE: scenarios and configurations

Giovanni Nardini, Antonio Virdis, Giovanni Stea
Dipartimento di Ingegneria dell'Informazione, University of Pisa
Largo Lucio Lazzarino 1, I-56122, Pisa, Italy
g.nardini@ing.unipi.it, a.virdis@iet.unipi.it, giovanni.stea@unipi.it

***Abstract*—SimuLTE is a tool that enables system-level simulations of LTE/LTE-Advanced networks within OMNeT++. It is designed such that it can be plugged within network elements as an additional Network Interface Card (NIC) to those already provided by the INET framework (e.g. Wi-Fi). Recently, device-to-device (D2D) technology has been widely studied by the research community, as a mechanism to allow direct communications between devices of a LTE cellular network. In this work, we present how SimuLTE can be employed to simulate both one-to-one and one-to-many D2D communications, so that the latter can be exploited as a new communication opportunity in several research fields, like vehicular networks, IoT and machine-to-machine (M2M) applications.**

Keywords—LTE-Advanced; simulation; SimuLTE; device-to-device

I. Introduction

In recent years, OMNeT++ [1] has become the reference framework for performing simulation, especially in the networking field. Several OMNeT++-based models are available to support researchers in assessing the performance of networking systems, e.g. INET [2]. SimuLTE is one of those models, which provides a simulation framework for the data plane of LTE-Advanced (LTE-A) networks [3]. A peculiarity of SimuLTE is its Network Interface Card (NIC) implementation: similarly to other models available in the INET framework (e.g. Wi-Fi) it can be easily integrated within any other OMNeT++ module, so as to endow nodes with LTE connectivity. It also embeds the possibility to simulate device-to-device (D2D) communications. D2D is a new communication paradigm that enables proximity services for cellular users, since it allows two or more terminals to communicate directly, without using the base station as a relay. In particular, both one-to-one and one-to-many D2D communications can support new services for a variety of scenarios, such as vehicular networks, Internet of Things (IoT) or machine-to-machine (M2M) applications [7]. For this reason, it has attracted increasing attention from the research community. Link-level simulations are typically used for testing channel properties of D2D links, also due to the substantially lack of tools for evaluating system-level performance of D2D. In this regard, SimuLTE enhanced with D2D may play a crucial role in allowing researchers to assess the performance of new algorithms and services in several scenarios and configurations.

In this paper, we describe how to setup simulations with D2D in SimuLTE, with a special focus on the main parameters that might affect the performance of the system. We also discuss how different settings of those parameters can be employed to evaluate different scenarios.

The remainder of the paper is organized as follows. An overview on D2D technology in LTE-Advanced networks is given in Section II. Section III presents the architecture of SimuLTE, enhanced to support D2D functionalities. Section IV describes the main parameters and the possible configurations for simulating one-to-one and one-to-many D2D communications. Section V concludes the paper.

II. Device-to-device communications in LTE

With reference to Figure 1, in a conventional LTE-Advanced network, every communication from/to cellular devices, called User Equipments (UEs), occurs only with the serving base station, which is called eNodeB (eNB), in either the downlink (DL) or the uplink (UL) direction. Recently, device-to-device (D2D) communications have been introduced in the LTE-Advanced standard [4]. D2D allows proximate UEs to exchange data directly using the so-called *sidelink* (SL) path, avoiding the two-hop path through the eNB, hence reducing latency and saving resources. Both *one-to-one* (or unicast) and *one-to-many* (or multicast) D2D communications are envisaged. In the one-to-one case, there is a peering session established between two UEs, which communicate in the same way as they would with the eNB. In the one-to-many case, the sender UE transmits data using a MAC-level identifier for the group of UEs that should receive the message. However, in both cases, the eNB still controls the allocation of resources, in either a (semi-)static or a dynamic way (e.g. assigning resource grants to UEs upon reception of a scheduling request). Hybrid Automatic Repeat reQuest (H-ARQ) feedback and retransmissions are supported for the unicast case only.

Both unicast and multicast D2D communications present peculiar challenges that are currently being studied by the research community, such as resource allocation. System-level simulations aid researchers to investigate potential solutions to those issues, hence we provide a framework that enables the simulation of D2D-capable devices in different scenarios and under different settings of the system parameters.

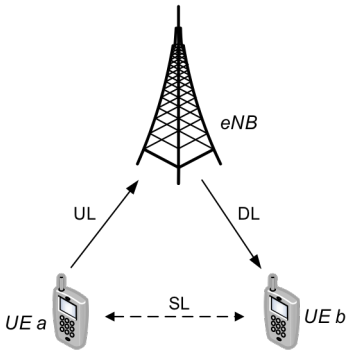


Fig. 1. Device-to-device communications

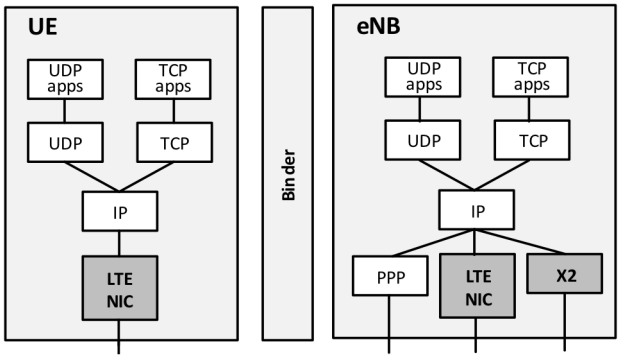


Fig. 2. SimuLTE architecture

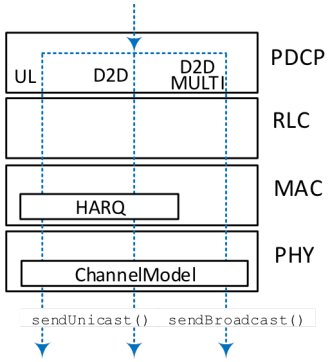


Fig. 3. Data flow within the sender UE's protocol stack

III. The SimuLTE Framework

This section provides a background on the architecture of SimuLTE, a system-level simulator based on the OMNeT++ and INET frameworks. The basic concept of OMNeT++ is *modularity*: simulations are built by composing different modules that interact with each other using messages. INET provides a large amount of entities and protocols for simulating both wired and wireless networks. In particular, it provides the concept of Network Interface Card (NIC) modules. The latter can be integrated within other modules so as to model different communication protocols between network devices. For example, devices can be endowed with Wi-Fi or Point-to-Point Protocol (PPP) NIC modules, or both. SimuLTE exploits this feature, as it is designed as an extension of a wireless NIC module. This allows one to add LTE capabilities to a node included in the simulation.

SimuLTE provides models for both UEs and eNB. As shown in Figure 2, both nodes contain the LTE NIC, together with modules implementing upper layer protocols, taken from INET. In addition, the eNB has an interface to the Internet via PPP and can also be connected to other eNBs using the X2 interface. The LTE NIC in both the UE and the eNB implements the whole LTE protocol stack, as one submodule per layer, namely Packet Data Convergence Protocol (PDCP), Radio Link Control (RLC), MAC and PHY. Since UEs and eNBs perform different operations within the protocol stack, SimuLTE exploits the inheritance paradigm of OMNeT++ for defining both the structure and behavior of each submodule. In particular, each submodule has common operations, which are extended with functionalities specific for the UE and the eNB, respectively. Communication between different layers occurs via message exchange, same as data transmission between UEs and eNBs. On the other hand, resource accounting is decoupled from data transmission. A dedicated module, called *Binder*, monitors which resources, i.e. Resource Blocks (RBs), are used by both the eNBs (for downlink transmissions) and the UEs (for uplink transmissions). The Binder can be considered as the oracle of the LTE network, since all the LTE nodes can access it to share common information via direct method calls.

Air transmissions between LTE NICs are modeled by the *ChannelModel* class, included within the PHY layer of the LTE NIC itself. On reception of a new message, the ChannelModel computes the Signal-to-Interference-and-Noise Ratio (SINR) perceived by the node. To do this, it obtains information from the Binder about the usage of the RBs for all the nodes in the network and decides whether the message can be successfully decoded or not. ChannelModel is also responsible for computing and reporting the Channel Quality Indicator (CQI) of the UEs, which is used for scheduling operations at the eNB. SimuLTE comes with a realistic implementation of the ChannelModel that takes into account path loss and fading effects. However, it can be easily re-implemented to allow one to employ its own preferred model.

We enhanced SimuLTE so as to endow the LTE NIC with D2D capabilities, enabling both one-to-one [5] and one-to-many direct communications between UEs. From the UE's perspective, most D2D operations are common to those performed for traditional UL transmissions, thus we implemented a new module per LTE layer, which inherits common functionalities from UL ones and adds D2D-specific functionalities, e.g. reporting of CQI on the SL path. With reference to Figure 3, data flows are forked at the PDCP level when they arrive from upper layers, based on whether the transmission direction is UL, D2D or D2D_MULTI. In the latter case, each packet contains also an identifier for the multicast group that must receive the packet. At the lower layers, each flow is subjected to different elaborations according to its direction. For example, in the multicast case, packets cannot use H-ARQ functionalities. When the packet reaches the PHY layer, a message is sent to the receiving UE using a `sendDirect()` call. As for the multicast case, the `sendBroadcast()` function sends a copy of the message to all UEs within the transmission range of the sender, exploiting the implementation of wireless broadcast transmissions of INET. At the reception side, UEs discard the message if they are not subscribed to the relevant multicast group.

IV. Simulating D2D with SimuLTE

SimuLTE natively provides models of UEs and eNBs, hence it is possible to simulate D2D communications within

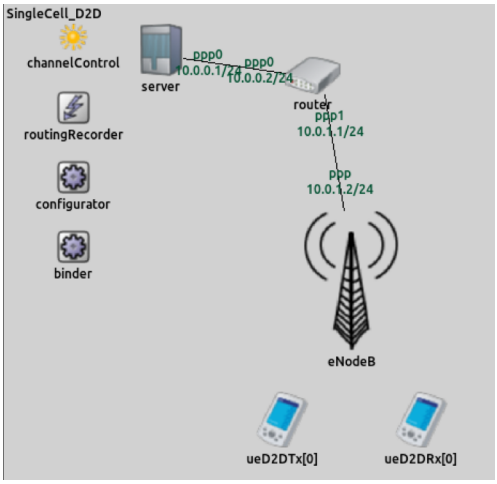


Fig. 4. Example scenario for one-to-one D2D communications

cellular networks, possibly mixed with other Internet devices, e.g. taken from INET. However, the NIC implementation of SimuLTE described in the previous section can be fit to any network device. For example, *VeinsLTE* [6] uses SimuLTE to simulate heterogeneous vehicular networks, where vehicles are equipped with both Wi-Fi and LTE NICs. Following this direction, it is possible to design nodes with multiple network interfaces, e.g. LTE, Wi-Fi, Bluetooth etc., and compare D2D with other technologies for short-range communications.

We now describe how to configure D2D-related simulation parameters in the `.ini` configuration file and how different parameters can be employed for evaluating different scenarios.

A. One-to-one D2D

Figure 4 reports an example scenario for the unicast case, where `ueD2DTx[0]` and `ueD2DRx[0]` are the sender and the receiver of the flow, respectively. A snippet of the `omnetpp.ini` file is reported below.

```
# enable D2D capabilities
*.eNodeB.d2dCapable = true
*.ueD2D*[*].d2dCapable = true

# select the AMC mode
*.eNodeB.nic.mac.amcMode = "D2D"

# set peering relationship
*.ueD2DTx[0].nic.d2dPeerAddresses="ueD2DRx[0]"

# select the CQI for D2D transmissions
*.eNodeB.nic.phy.enableD2DCqiReporting = false
**.usePreconfiguredTxParams = true
**.d2dCqi = 7

# set Tx Power
*.ueD2DTx[0].nic.phy.ueTxPower = 26   # in dB
*.ueD2DTx[0].nic.phy.d2dTxPower = 20  # in dB
```

In order to enable D2D communication between the two UEs, they must be defined as D2D-capable UEs, as well as the eNB. To do this, the `d2dCapable` parameter of the involved nodes has to be set in the `.ini` file. For the eNB, it is necessary to use the D2D-specific Adaptive Modulation and Coding (AMC) module, for computing transmission parameters for the SL. Then, we need to configure the receiver as a possible D2D peer for the sender. To this aim, the `d2dPeerAddresses` parameter is a blank spaces-separated list of IP addresses or aliases of UEs. Note that peering is unidirectional, hence we need to explicitly define the reverse one for bidirectional flows, e.g. TCP connections.

As far as CQI computation is concerned, it is possible to use fixed CQIs for every D2D transmission, set at the beginning of the simulation. Otherwise, it is possible to let sender UEs report the CQI measured on the D2D links, for all their peering UEs, i.e. those in the `d2dPeerAddresses` list. The latter mode allows one to obtain link adaptation of the SL, as for UL and DL communications. On the other hand, this method is computationally heavier, since CQI reporting may be done periodically and the number of peering UEs may be high. In the example we use fixed CQI, thus we set the `usePreconfiguredTxParams` parameter and define the CQI to be used, i.e. `**.d2dCqi=7`. In this case, the `enableD2DCqiReporting` parameter is set to `false`. If both modes are enabled, preconfigured CQI would be used, and reporting would be useless and would only waste computations. At the PHY layer, it is possible to employ different transmission power for UL and D2D. To do this, `ueTxPower` and `d2dTxPower` parameters should be set accordingly. This allows one to evaluate, for example, the maximum distance or the consumed energy for a direct communication under specific combinations of CQI and transmission power. Since D2D enables spatial frequency reuse, different settings also affects interference among UEs allocated in the same resources, hence it is possible to assess the performance of scheduling algorithms that exploit frequency reuse and the coexistence between D2D and traditional infrastructure communications.

The communication mode between two D2D-capable UEs may be changed during the simulation, from D2D mode to the traditional infrastructure mode. In fact, UEs may get too far from each other, thus reducing the SINR on the SL, or the eNB may want to optimize the allocation of resources according to its own policy. This topic has recently gained attention in the research community, e.g. [8][9], and the need to simulate those algorithms may arise. To this aim, we endow the LTE NIC at the eNB with a module called `d2dModeSelection`. It is implemented as an abstract class that periodically invokes the `doModeSelection()` function. The latter runs an algorithm that, for each D2D-capable UEs, selects which mode has to be be used in the next period to communicate with each of its possible receivers, i.e. those in the `d2dPeerAddresses` list. Since `doModeSelection()` is a pure virtual function, it must be re-implemented by derived classes, as shown in Figure 5. The `d2dModeSelectionBestCqi` module implements a simple algorithm that chooses either D2D or infrastructure mode based on the best CQI between the UL and the SL. This architecture allows one to realize its own mode selection policy by extending the base module and implementing the

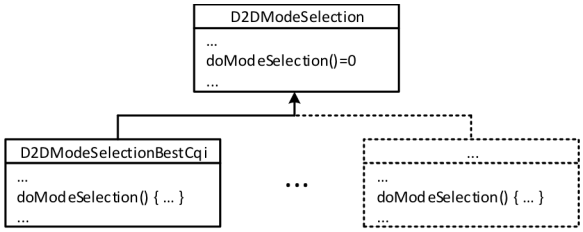


Fig. 5. D2D Mode Selection architecture

`doModeSelection()` function. When the selection has been made for each possible D2D pair, a control message is sent to both the endpoints to command the switch to the new communication mode. This operation is similar to the handover and ongoing communications between the UEs involved in the mode switch may experience packet loss if no countermeasures are taken. Thus, it is possible to investigate new architectural solutions to improve the performance of this operation [10]. To enable dynamic mode switching, the following lines must be added to the `omnetpp.ini`, where the `d2dModeSelectionType` parameter contains the name of the module implementing the mode selection algorithm.

```
*.eNodeB.nic.d2dModeSelection = true
*.eNodeB.nic.d2dModeSelectionType="D2DModeSelectionBestCqi"
```

B. *One-to-many D2D*

With reference to the scenario of Figure 6, we consider one sender UE, `ueD2D[0]` and two receiving UEs, namely `ueD2D[1]` and `ueD2D[2]`. Multicast messages are generated at the application layer of the node and sent towards a multicast IP address. Thus, they can be received only by UEs that subscribed to the addressed IP multicast group. The latter has to be defined within the XML configuration file of the `IPv4NetworkConfigurator` module. In the example of Figure 6, this is accomplished as follows, where all UEs belong to the group having address 224.0.0.10.

```
<multicast-group hosts="ueD2D[*]"
    interfaces="wlan" address="224.0.0.10"/>
```

In order to let `ueD2D[0]` send multicast messages, the destination IP address of packets generated by its application layer must be set to that value in `omnetpp.ini` file.

```
*.ueD2D[0].udpApp[*].destAddress = "224.0.0.10"
```

The configuration of the `omnetpp.ini` file presents just few changes w.r.t. the one-to-one case described before. However, for one-to-many communications, CQI can be configured only in fixed mode, i.e. setting the `usePreconfiguredTxParams`. The selected CQI affects the transmission range of the communication. In fact, smaller CQIs allow the transmission to be received at farther distances at the cost of higher resource consumption, as more RBs are required to send a message. This can be exploited, for example, in vehicular networks for broadcasting traffic and/or collision information, even in multihop scenarios. In this case, different CQIs allow the packet to reach a different number of

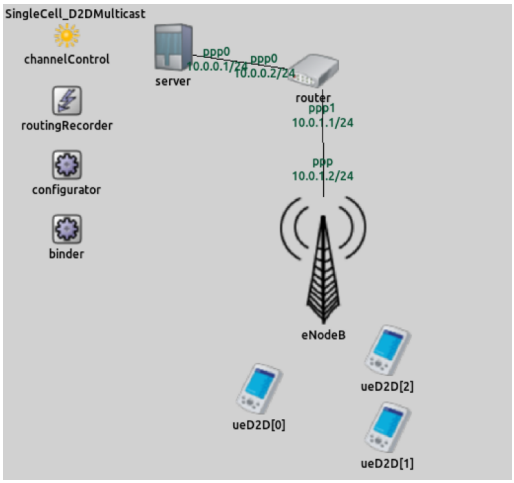


Fig. 6. Example scenario for one-to-many D2D communications

vehicles, affecting the performance of the dissemination in terms of latency and resource consumption [11].

V. Conclusions

In this work, we presented SimuLTE as a simulation tool for D2D communications in LTE-Advanced networks. In particular, we provided insight on how to configure simulation parameters to assess the impact of different factors on D2D systems. The presented framework can be exploited, for example, to compare the performance of D2D with other wireless technologies, like Wi-Fi, or to evaluate any kind of network system (e.g. vehicular networks) that might benefit from using one-to-one or one-to-many D2D communications.

References

[1] A. Varga, R. Hornig, "An overview of the OMNeT++ simulation environment", in Proc. SIMUTools '08, Marseille, FR, March 2008.

[2] The INET framework, https://inet.omnetpp.org/

[3] A. Virdis, G. Stea, G. Nardini, "Simulating LTE/LTE-Advanced Networks with SimuLTE", DOI 10.1007/978-3-319-26470-7_5, in: Advances in Intelligent Systems and Computing, Vol. 402, pp. 83-105, Springer, 15 January 2016.

[4] 3GPP - TS 36.843 v12.0.1, "Study on LTE Device-to-device Proximity Services: Radio aspects (Release 12)", March 2014.

[5] A. Virdis, G. Nardini, G. Stea, "Modeling unicast device-to-device communications with SimuLTE", IWSLS2 2016, Vienna, July 1st, 2016.

[6] F. Hagenauer, F. Dressler, C. Sommer, "A Simulator for Heterogeneous Vehicular Networks," IEEE VNC 2014, Paderborn, DE, Dec 2014.

[7] A. Asadi, Q. Wang, V. Mancuso, "A survey on device-to-device communication in cellular networks," IEEE Comm. Surveys and Tutorials, vol.16, no.4, pp.1801-1819, Fourth quarter 2014.

[8] G. Nardini, G. Stea, A. Virdis, D. Sabella, M. Caretti, "Resource allocation for network-controlled device-to-device communications in LTE-Advanced", Wireless Networks, 2017, DOI: 10.1007/s11276-016-1193-3.

[9] K. Doppler, Chia-Hao Yu, C.B. Ribeiro, P. Janis, "Mode Selection for Device-To-Device Communication Underlaying an LTE-Advanced Network," Proc. of WCNC 2010, pp.1-6, 18-21 April 2010.

[10] G. Nardini, G. Stea, A. Virdis, D. Sabella, M. Caretti, "Fast and agile lossless mode switching for D2D communications in LTE-Advanced networks", IEEE VTC Spring 2016, Nanjing, PRC, 15-18 May 2016.

[11] G. Nardini, G. Stea, A. Virdis, D. Sabella, M. Caretti, "Broadcasting in LTE-Advanced networks using multihop D2D communications", PIMRC 2016, Valencia, September 5-7, 2016.

Extending OMNeT++ Towards a Platform for the Design of Future In-Vehicle Network Architectures

Till Steinbach, Philipp Meyer, Stefan Buschmann, and Franz Korf
Department of Computer Science, Hamburg University of Applied Sciences, Germany
{till.steinbach, philipp.meyer, stefan.buschmann, franz.korf}@haw-hamburg.de

***Abstract*—In-vehicle communication technologies are evolving. While today's cars are equipped with fieldbusses to interconnect the various electronic control units, next generation vehicles have timing and bandwidth requirements that exceed the capacities. In particular Advanced Driver Assistance Systems (ADAS) and automated driving using high bandwidth sensors such as cameras, LIDAR or radar will challenge the in-car network. Automotive Ethernet is the most promising candidate to solve the upcoming challenges. But to design and evaluate new protocols, concepts, and architectures suitable analysis tools are required. Especially in the interim period with architectures using automotive Ethernet and legacy fieldbusses together, careful planning and design is of vital importance. Simulation can provide a good understanding of the expectable network metrics in an early development phase.**

This paper contributes a workflow as well as the required toolchain to evaluate new real-time Ethernet communication architectures using event based simulation in OMNeT++. We introduce a domain specific language (DSL) – the Abstract Network Description Language (ANDL) – to describe and configure the simulation and present the required simulation models for real-time Ethernet and fieldbus technologies such as CAN and FlexRay. We further introduce new analysis tools for special in-vehicle network use-cases and the interaction of the simulation with third-party applications established in the automotive domain.

***Index Terms*—Simulation, In-Vehicle Networking, Real-time Ethernet, Automotive Ethernet, CAN, FlexRay, Gateway**

I. Introduction & Problem Statement

The in-vehicle network faces a significant paradigm change. While communication architectures of today's vehicles are consisting of different technologies such as Controller Area Network (CAN), FlexRay, Local Interconnect Network (LIN) and Media Oriented Systems Transport (MOST), soon Ethernet will form the backbone for in-vehicle communication. Switched Ethernet is – due to its high data rate, its low cost of commodity components, and its large flexibility in terms of protocols and topologies – a promising candidate to overcome the challenges of future in-car networking [1]. But, a sudden change from today's architectures towards a network solely build on Ethernet is impossible with reasonable cost and risk. A consolidation strategy with heterogeneous networks formed of an Ethernet root and legacy busses at the edges will allow to preserve invest in knowledge around legacy technologies. Such a mixed architecture can form the beginning of a stepwise transition from today's bus based designs towards a flat network topology using Ethernet links only.

To design and evaluate such future in-vehicle networks, new tools are required. While current toolchains focus on bit-correct simulation of fieldbus communication, future environments have to enable the developer to analyze effects of congestion and jitter on the cars applications and assistance functions on a system level. The OMNeT++ [2] platform is a well suited tool and a perfect base to implement a flexible workflow. Besides its open-source simulation core, it allows to extend its Eclipse based IDE with custom plugins for specialized design and analysis tasks. With this work we contribute both, a uniform workflow as well as the required models and tools to design and evaluate future in-vehicle networks.

The center of the simulation toolchain are the simulation models that are published open-source. For various real-time Ethernet technologies the CoRE4INET model suite was created. It relies on the OMNeT++ INET framework and provides the implementation of real-time Ethernet protocols as well as clock synchronization. The models for fieldbus technologies are similarly provided in the FiCo4OMNeT suite. To interconnect both technologies – real-time Ethernet and fieldbusses – the SignalsAndGateways models were developed.

Experiences with the simulation during research on in-car network architectures showed that the configuration of these large networks is complex and lengthy. Thus there was a demand to simplify the description of in-car network scenarios. This demand led to the development of a domain specific language (DSL) that supports the fast setup of simulations of in-car network architectures.

Finally, in-car networks require the analysis of specific network metrics, for example tracing of jitter in the forwarding chain of cyclic messages. To support the evaluation of in-car networks we created analysis tools and interfaces that support the workflow and allow to pass simulation results to third-party software. While some of these tools are specific to in-car networking, most of the software provided for result analysis is applicable to other network simulation scenarios as well.

The remaining paper is organized as follows: In Section II, we introduce the technological background and relate to preliminary and related work. Section III presents the simulation models for in-vehicle networks. We present tools for designing and configuring in-vehicle networks in section IV and tools to analyze the simulation results in section V. Using a short case study, section VI presents our workflow. Finally, section VII concludes our work and gives an outlook on future research.

II. Background & Related Work

Today there are several commercial tools to analyze in-car networks. In industry, most popular is CANoe (by Vector Informatik GmbH) that enables real-time cluster simulations of fieldbusses. Today, CANoe does not provide functionality to simulate real-time Ethernet variants. SymTA/S is a commercial timing analyzer (not a network simulator) by Symtavision GmbH that supports Ethernet (standard and AVB) as well as common fieldbus technologies. It provides analytical models to calculate load and timing.

OMNeT++ [2] is a discrete event based simulation platform mainly focusing on the simulation of networks and multi-processor systems. It is a perfect base for a simulation tool chain for automotive communication. We developed our model suites as an extension of the popular INET-Framework [3] that provides the implementation of Ethernets physical layer as well as protocols and applications above layer 2.

While there were no publicly available OMNeT++ simulation models for real-time Ethernet technologies, there is another CAN bus model developed independently at the same time at the Nagoya University in Japan. The last release is from April 2014 and is not yet compatible with the latest INET and OMNeT++ releases. The developers also analyzed CAN-Ethernet gateway strategies [4].

For collecting simulation results in databases there are already examples for MySQL provided with OMNeT++. A simple interface for storing results in SQLite is provided with the INET-HNRL, a fork of the INET framework for hybrid networking research [5].

III. Simulation Models

The simulation models introduced in this section were developed for in-car network simulations, but can be used for other systems as well. All models are published open-source (see http://sim.core-rg.de) and can be used free of charge. Figure 1 gives an overview of the contributed simulation models and their place in the software stack of the toolchain. To simplify the installation an OMNeT++ plugin is provided that offers an automated installation process as well as an update procedure.

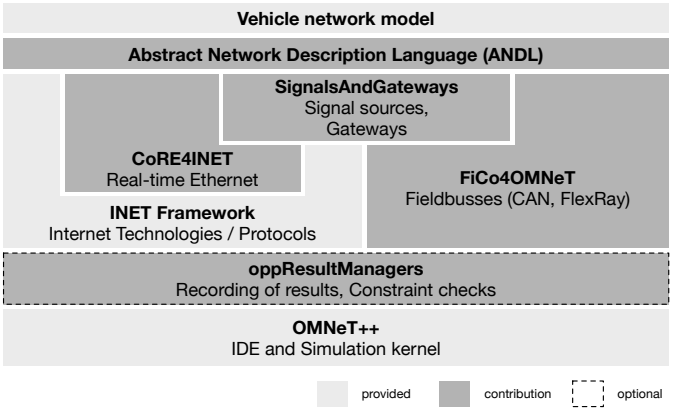


Fig. 1. Overview of the contributed tools and simulation models

A. CoRE4INET

CoRE4INET (Communication over Real-time Ethernet for INET) is a suite of real-time Ethernet simulation models. Currently it supports the AS6802 protocol suite, traffic shapers of Ethernet AVB, and implementations of IEEE 802.1Q, as well as models to map IP traffic to real-time traffic classes.

The center of the CoRE4INET models is the implementation of media access strategies for different traffic classes. By flexibly combining these strategies, new traffic shapers can be designed that are able to forward real-time traffic of different standards. For example it is possible to combine time-triggered traffic of AS6802 with credit based shaping of Ethernet AVB to form a new time-aware shaper that can handle both classes in parallel [6]. This allows to evaluate new concepts that are currently under standardization or are even not yet assessed.

For incoming traffic, the models contain traffic selection and constraint checks. To simulate time-triggered behavior and time-synchronization, CoRE4INET provides models for oscillators, timers and schedulers. Oscillators allow to implement the behavior of inaccurate clocks with their unique influence on real-time communication. Finally, CoRE4INET contains application models for simple traffic patterns and traffic bursts.

The simulation models were checked against analytical models of the different specifications and – where possible – evaluated in empirical tests using real-world hardware.

CoRE4INET is under active development. New features are constantly added. The next milestone targeting for release soon is the support of frame-preemption as currently discussed in IEEE PAR 802.1Qbu [7].

B. FiCo4OMNeT

FiCo4OMNeT (Fieldbus Communication for OMNeT++) is a set of fieldbus simulation models. It was originally developed with separate CAN and FlexRay models, but later merged to take account for the similarities of fieldbus technologies.

Similar to CoRE4INET, FiCo4OMNeT contains models of oscillators and clocks to allow for a precise simulation of the synchronization of FlexRay's static segment. Further it contains application models for CAN and FlexRay applications with simple traffic patterns. The fieldbus models in FiCo4OMNeT were originally checked against results of the CANoe simulation environment, an industry standard software for the simulation of fieldbus based in-vehicle networks.

Even though fieldbusses are considered legacy technology there are new approaches. With CAN-FD (CAN with flexible data rate) the bandwidth of CAN can be significantly increased. We are currently working on a CAN-FD implementation to enable the simulation of advanced in-car network architectures containing CAN busses with flexible datarate.

C. SignalsAndGateways

The SignalsAndGateways simulation models fill the gap between the simulation of real-time Ethernet and fieldbusses. Gateways are nodes that translate between legacy bus technologies and (real-time) Ethernet. To be as flexible as possible, the gateways are divided in three submodules:

a) Routing: The router module receives messages in their original representation and decides based on forwarding rules which path the message will take. A message can have no routing entry if it should be dropped, one routing entry if it has one receiving bus or node, or several routing entries if it should be visible to several receivers on different busses. There is no limit of busses and links a gateway can be connected to. The gateway can also translate between fieldbus technologies, thus it is also applicable to legacy designs with multiple busses interconnected over a central gateway.

b) Buffering: Gateways support aggregation strategies to improve bandwidth utilization of different technologies. CAN messages for example have a maximum payload of 8 B, while Ethernet messages have a minimum payload of 46 B. If only one CAN message would be encapsulated in an Ethernet frame, the rest of the frames payload would be padded and bandwidth would be wasted. Aggregation strategies implemented in the buffer modules allow to release frames in groups, according to different strategies. These strategies are implemented in the buffer modules, too.

Aggregation strategies have a huge impact on the latency of messages passing a gateway. All strategies delay frames to collect multiple messages before aggregating them into one large packet. The most popular strategy implemented in the buffer is the pooling strategy with holdup time. Each message is assigned to a pool, while multiple different messages share the same pool. Further each message is assigned a holdup time, representing the maximum acceptable delay for this message. On arrival of a frame in the pool, its holdup time is compared with the pools holdup time. If the frames holdup time is shorter, the pools time is adjusted correspondingly. When the holdup time of the pool is expired all messages in the pool are released together. The modular architecture of the gateway allows to easily add more aggregation strategies.

c) Transformation: Transformation modules implement the translation between different communication technologies. The strategies transparently map information between fieldbusses and Ethernet. Currently there is a simple mapping between fieldbus frames and raw (layer 2) Ethernet frames. The modular architecture of the gateway allows to easily add more sophisticated mappings, e.g. when higher layer application protocols should be used.

Similar to real-world gateways, gateway nodes can host applications that are not related to gateway functionality. Thus gateways can be added to control units that also host application software.

IV. Network Design

Configuring the simulation of large heterogeneous networks is complex and lengthy. To reduce this effort and to let the developer focus on his design task, we developed a domain specific language (DSL) for the description of heterogeneous in-vehicle network designs. It is called *Abstract Network Description Language (ANDL)* and provides an easy and assisted way to design a network in an Eclipse environment. It is implemented as an Eclipse plugin and thus fits into the OMNeT++ IDE. The plugin provides syntax highlighting as well as context aware code completion. For TDMA technologies, the ANDL plugin contains scheduling algorithms that allow to find a first feasible schedule for initial results [8].

Listing 1 shows an example of a network consisting of two CAN busses interconnected over a real-time Ethernet backbone described in the ANDL.

Listing 1
ANDL code example with comments

```
types std {                 //Types can be defined and reused
  ethernetLink ETH {        //Definition for Ethernet link
    bandwidth 100Mb/s;      //Link has bandwidth of 100MBit/s
  }
}   //it is also possible to define types in a separate file

network smallNetwork{ //network name is smallNetwork
  inline ini{           //Inline ini for special parameters
    record-eventlog = false
  }                     //Parameters are inserted into .ini

  devices{              //Define all devices in the network
    canLink bus1;       //First CAN bus
    canLink bus2;       //Second CAN bus
    node node1;         //First CAN node
    node node2;         //Second CAN node
    gateway gw1;        //Gateway for first CAN bus
    gateway gw2;        //Gateway for second CAN bus
    switch switch1;     //Real-time Ethernet Switch
  }

  connections{     //Physical connections (Segments = groups)
    segment backbone { //Ethernet Backbone part
      gw1 <--> {new std.ETH} <--> switch1; //Ethernet Link
      gw2 <--> {new std.ETH} <--> switch1; //Ethernet Link
    }
    segment canbus{     //CAN bus part (busses share config)
      node1 <--> bus1; //CAN node connected to first bus
      gw1 <--> bus1;   //Gateway connected to first bus
      node2 <--> bus2; //CAN node connected to second bus
      gw2 <--> bus2;   //Gateway connected to second bus
    }
  }

  communication{        //Communication in the network
    message msg1{       //Message definition
      sender node1;     //First CAN node is sender
      receivers node2; //Second CAN node is receiver
      payload 6B;       //Message payload is 6 Bytes
      period 5ms;       //5ms cyclic transmission
      mapping{  //mapping to traffic class, id, gw strategy
        canbus: can{id 37;};        //Message ID 37 on CAN
        backbone: tt{ctID 102;};    //TT traffic on backbone
        gw1: pool gw1_1{holdUp 10ms;};   //Aggregation time
      }
    }
  }
}
```

In comparison to the compact description in ANDL, the size of the generated OMNeT++ config (.ini/.ned/.xml) has more than 250 lines. The resulting network is shown in Figure 2. The definition of the scenario starts with the networks *devices*. Afterwards the previously defined devices are arranged into a network topology in the *connections* section. The topology can be divided in several different segments with different configurations for messages. In the example there is one segment for the Ethernet part called *backbone* and one segment for the CAN bus part called *canbus*. When messages traverse the borders of a segment they are translated from the sending segments representation into the receiving segments

Fig. 2. ANDL generated network consisting of two CAN busses and a real-time Ethernet backbone with two gateways and one switch

representation. The last part of the definition is the actual communication taking place. In the example there is only one message transmitted from *node1* to *node2*. The mapping of each message defines how the message is represented in the different segments. In the example the message is a CAN frame with id *37* on the bus and a time-triggered message with critical traffic id *102* on the real-time Ethernet backbone.

Besides the features shown, the ANDL defines more parameters to describe traffic flows or aggregation strategies. Commonly used components can be defined in include files, e.g. a Ethernet Link with $100\,\mathrm{Mbit/s}$, and used in several places. Further ANDL provides inheritance, thus it is possible to define primitive stencils for components that are later refined during the instantiation.

Currently, ANDL supports only the most commonly used parameters. For more sophisticated configurations inline ini code can be used. Parameters defined in the inline ini sections are directly copied into the resulting omnetpp.ini file.

The ANDL is implemented as an OMNeT++ plugin using Eclipse's Xtext technology. Xtext is a framework for development of programming languages and domain-specific languages. It provides a grammar to define the language and generates the required parsers as well as the code editor for the OMNeT++ IDE.

V. Result Analysis

The OMNeT++ IDE already comes with tools for the result analysis. We extended those built-in tools to simplify the analysis in specialized use-cases and developed interfaces to interconnect the OMNeT++ simulation with established industry products. Our contributions for the result analysis are:

A. Gantt Chart Timing Analyzer

The Gantt Chart Timing Analyzer (GCTA) is an OMNeT++ plugin developed to trace jitter and delay in cyclic communication. It uses a timing log file (.tlog) written during the simulation to generate a gantt chart of the communication path from sender to receiver. In contrast to the OMNeT++ eventlog, that shows all events of the simulation in a timeline, the GCTA plugin compresses all occurrences of a cyclic message into one single chart. This allows to easily detect the source of jitter and delay in the path between sender and receiver.

After installing the GCTA plugin in the OMNeT++ IDE, .tlog files (see section V-B3) recorded during the simulation can be processed. Currently, GCTA only supports analysis of (real-time) Ethernet traffic, but the concept is also applicable to heterogeneous networks with Ethernet and fieldbusses interconnected using gateways. GCTA is an example for a specialized analysis tool implemented as OMNeT++ plugin that extends the built in functionality. As it relies on the visualization capabilities of OMNeT++ no further software is required.

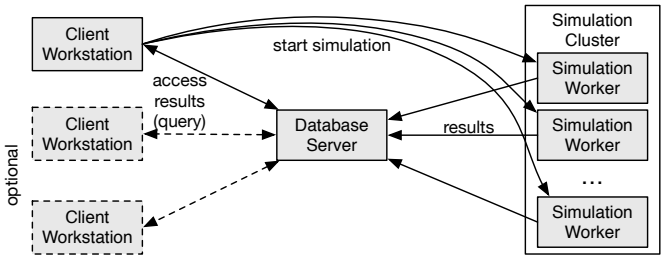


Fig. 3. Storing and accessing simulation results using a database (e.g. postgreSQL).

B. oppResultManagers

oppResultManagers is a set of modules for OMNeT++ simulations. Instead of simulation models it contains so called ResultManagers. ResultManagers are responsible for writing out simulation results. The OMNeT++ vector and scalar files, as well as the eventlog are built-in instances of ResultManagers. The oppResultManagers project adds the following ResultManagers to OMNeT++:

1) PCAPng: The PCAP next generation (PCAPng) dump file format [9] is an attempt to overcome the limitations of the currently widely used (but limited) libpcap format. Libpcap allows to log packet oriented communication and is used in popular analysis tools such as Wireshark. The most important extension of PCAPng is the support of multiple interfaces in one file. The INET framework already contains a module to write legacy PCAP files, but it only supports communication above IP layer. Due to the PCAPng module being a ResultManager, no changes to the simulation must be made to write PCAPng files, it is simply enabled in the ini-confiuration in OMNeT++:

```
eventlogmanager-class = "PCAPNGEventlogManager"
```

The PCAPng manager uses the packet serialization feature of the INET framework. Thus it is able to write packets for all protocols and layers as long as a serializer was previously implemented and registered. This makes it applicable in other domains, e.g. wireless communication, or for verifying the implementation of models of protocols in OMNeT++.

2) SQLite & postgreSQL: The SQLite and postgreSQL ResultManagers allow to store simulation results into a SQLite database file or a postgreSQL database. Databases allow to perform complex queries on the simulation results. This can significantly speedup the process of obtaining network metrics, especially when huge parameter sets with various seeds were simulated.

SQLite is a file based database that is fast and efficient. The SQLite ResultManager is in most cases slightly slower than OMNeT++ vector and scalar managers, but produces smaller result files. The SQLite database is always stored on the machine executing the simulation. As the database is locked when it is opened for writing (similar to a OMNeT++ result file) several simulation processes running concurrently cannot write to the same database. Thus different parameter sets or seeds simulated in parallel will have to use separate SQLite databases. A script provided with the result manager

offers to merge several databases to enable queries containing results of several runs. As SQLite databases are regular files on the filesystem they can be easily transferred, archived, or read and manipulated using third party software.

The postgreSQL manager allows to write simulation results on a central database server in the network, while simulations are executed on a distributed cluster of nodes (see Figure 3). Several users can access the results concurrently without the necessity to distribute the result files. This allows to transfer the load of the simulation as well as result analysis from the users workstations towards strong servers and large centralized storage systems. The drawback of this solution is a slight performance decrease due to the overhead of sending results over the network as well as delays due to the databases lock mechanisms when it is accessed concurrently. Using a database system, OMNeT++ simulations can be easily attached to a wide range of analysis tools, e.g. R using a database driver.

The database ResultManagers are enabled in the .ini configuration of the simmulation:

```
outputscalarmanager-class="cPostgreSQLOutputScalarManager"
outputvectormanager-class="cPostgreSQLOutputVectorManager"
postgresqloutputmanager-connection="dbname=testdb user=
    testuser password=testuser port=15432"
```

3) GCTA: Writes out the previously introduced .tlog files for the GCTA plugin (see section V-A). The .tlog files contain information about the simulated topology as well as timing of cyclic messages. Traffic flows are aggregated based on their traffic classes. For example for messages using AS6802 the messages are grouped using the virtual link id, correspondingly for Ethernet AVB the stream id is used.

4) Constraint Check: The ResultManager for constraint checks allows to define rules for output vectors. The defined bounds are not to be violated by the simulation. The ResultManager writes out a report of possible violations and can also end the simulation if a violation was detected. This way a parameter set with undesired results won't unnecessarily utilize CPU resources. For example, if a large number of simulations with different parameters and seeds is being batch executed, a run that does not comply with the requirements is immediately stopped when the violation occurs. This can significantly reduce the time required to simulate large sets of parameters.

Currently constraints include minimum and maximum checks, minimum and maximum checks using an average over a number of samples, over a time interval and checks using the sum of a vector. Constraints are configured in a XML format (see listing 2) and can be easily extended.

Listing 2
EXAMPLE OF XML DEFINITION FOR THE CONSTRAINT CHECK RESULTMANAGER

```
<constraints>
  <constraint module="Network.node1"
      name="rxMessageAge:vector">
    <min>1.5</min>
    <max>1.7</max>
  </constraint>
  <constraint module="(.*)\.node2"
    moduleIsRegex="true"
    name="(rx|tx)MessageAge:vector"
    nameIsRegex="true">
    <avg_min samples="10">1.5</avg_min>
    <avg_max samples="10">1.7</avg_max>
  </constraint>
</constraints>
```

5) Multiple: This ResultManager allows to use several managers in parallel, enabling the user to write e.g. the legacy vector files in parallel with one of the new formats previously presented.

VI. CASE STUDY

The simulation environment is a valuable and established part of our daily research and development.

A. Simulation Workflow

Our workflow (see figure 4) starts with the network design. The ANDL is used to describe the required nodes, as well as the desired network topology, and the mapping of messages to different traffic classes. Afterwards our toolchain automatically generates an executable simulation configuration that is run using the simulation models for real-time Ethernet and field-busses. After the simulation run, the results are analyzed with the various result analyzers that are built into the OMNeT++ IDE, provided as additional plugins (e.g. the GCTA), or interconnected using databases and specialized output formats such as PCAPng.

B. Simulating the Ethernet Backbone of a Prototype Car

The presented simulation environment is used in several of our publications and is a base for the evaluation of new architecture concepts for in-car networks. So far, our largest project is the simulation of network designs for a real-world prototype car [10].

The simulated prototype is a Volkswagen Golf 7 that was equipped with a real-time Ethernet backbone for the RECBAR research project. The car originally contains seven domain specific CAN busses interconnected over a central gateway node. For the prototype a real-time Ethernet backbone using three real-time switches and several additional nodes with high

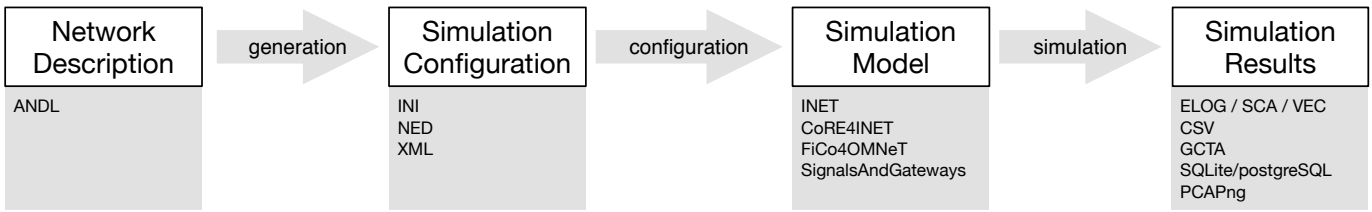


Fig. 4. Workflow of simulation projects – From network description to result analysis

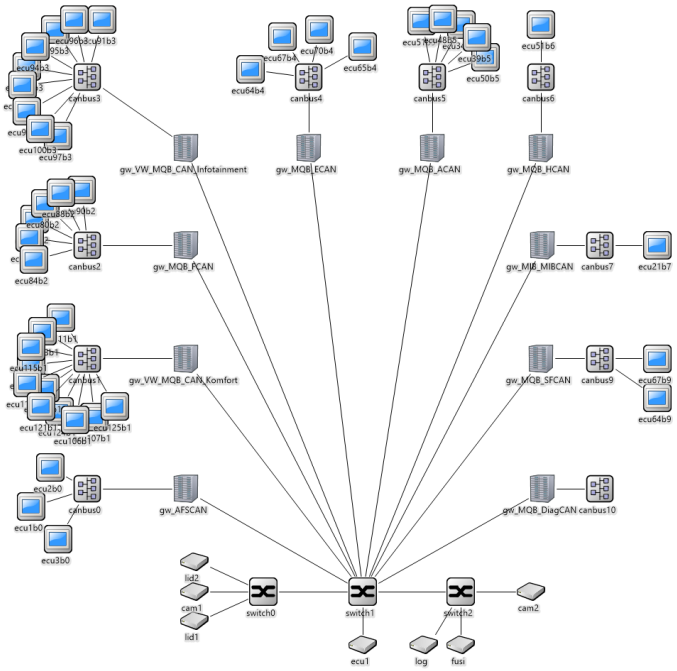


Fig. 5. Simulation of the Architecture of the RECBAR prototype

bandwidth applications such as high definition cameras and laser scanners were added. We simulate the prototype with real traffic patterns of the series car. Figure 5 shows the network.

In the simulation we are able to compare the results of the legacy network containg a central gateway node, with the results from the real-time Ethernet backbone. The results show that the real-time Ethernet solution can provide comparable end-to-end latency and jitter, while providing significant bandwidth reserves. The simulation allows us to evaluate the influences of different gateway aggregation strategies (see section III-C) and additional best-effort background cross-traffic.

Compared to empirical measurements in the real-world prototype, the simulation allows faster assessment of a wide range of protocols and configuration parameters. Further, the transparent nature of the network simulation allows us to debug configuration errors much faster than using the real system. While real hard- and software requires us to apply probes and adopt code to trace errors and measure timing, the simulation already provides thousands of measuring points.

VII. Conclusion & Outlook

In-vehicle communication technologies are about to change from todays fieldbusses to switched real-time networks. With network simulation on system-level the process of evaluating real-time and application protocols, developing new shaping strategies, assessing architectures, and predicting hardware requirements can be supported. We contribute a simulation environment consisting of simulation models as well as development and analysis tools to face the challenges in this upcoming technology transition. Our experiences with the simulation of in-car networks for prototypes and complex heterogeneous network architectures for future cars underlines the value of a uniform system-level simulation environment.

Our experiences with the development of OMNeT++ plugins and ResultManagers specialized for tasks in our domain of in-car network research show that for the daily work in development and research projects it is worth analyzing whether specialized tools can support the simulation workflow. With its Eclipse based IDE, OMNeT++ is a solid foundation for the development of such tools.

In our future work we focus on adding new technologies to our simulation suite, such as Ethernet with frame preemption as currently discussed in IEEE 802.1Qbu or the implementation of CAN with flexible data rate (CAN FD). We further work on refining our result analysis tools.

Download

All simulation models as well as the analysis tools presented in this work are published open-source and can be downloaded from our website at:

http://sim.core-rg.de

We further provide an Eclipse update site to simplify the installation of the presented OMNeT++ plugins.

Acknowledgements

This work was supported by the German Federal Ministry of Education and Research (BMBF) under the project RECBAR.

References

[1] K. Matheus and T. Königseder, *Automotive Ethernet*. Cambridge, United Kingdom: Cambridge University Press, Jan. 2015.

[2] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, networks and systems & workshops*. New York: ACM-DL, Mar. 2008, pp. 60:1–60:10.

[3] OMNeT++ Community, "INET Framework for OMNeT++ 5.0." [Online]. Available: http://inet.omnetpp.org/

[4] K. Kawahara, Y. Matsubara, and H. Takada, "A Simulation Environment and preliminary evaluation for Automotive CAN-Ethernet AVB Networks," in *Proceedings of the 1st OMNeT++ Community Summit, Hamburg, Germany, September 2, 2014*, A. Förster, C. Sommer, T. Steinbach, and M. Wählisch, Eds. ArXiv e-prints, Aug. 2014.

[5] K. S. Kim, "INET-HNRL Fork of INET Framework for Hybrid Networking Research." [Online]. Available: https://github.com/kyeongsoo/inet-hnrl

[6] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt, "Extending IEEE 802.1 AVB with Time-triggered Scheduling: A Simulation Study of the Coexistence of Synchronous and Asynchronous Traffic," in *2013 IEEE Vehicular Networking Conference (VNC)*. Piscataway, New Jersey: IEEE Press, Dec. 2013, pp. 47–54.

[7] IEEE 802.1 TSN Task Group, "IEEE 802.1Qbu - Frame Preemption." [Online]. Available: http://www.ieee802.org/1/pages/802.1bu.html

[8] J. Kamieth, T. Steinbach, F. Korf, and T. C. Schmidt, "Design of TDMA-based In-Car Networks: Applying Multiprocessor Scheduling Strategies on Time-triggered Switched Ethernet Communication," in *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2014)*. Piscataway, New Jersey: IEEE Press, 2014, pp. 1–9.

[9] M. Tuexen, F. Risso, J. Bongertz, and G. Harris, "PCAP Next Generation (PCAPNG) Dump File Format," Working Draft, IETF Secretariat, Internet-Draft draft-tuexen-opswg-pcapng-00, June 2014. [Online]. Available: http://www.ietf.org/internet-drafts/draft-tuexen-opswg-pcapng-00.txt

[10] T. Steinbach, K. Müller, F. Korf, and R. Röllig, "Real-time Ethernet In-Car Backbones: First Insights into an Automotive Prototype," in *2014 IEEE Vehicular Networking Conference (VNC)*. Piscataway, New Jersey: IEEE Press, Dec. 2014, pp. 137–138.

Simulation of the IEEE 1588 Precision Time Protocol in OMNeT++

Wolfgang Wallner
Vienna University of Technology
Email: wolfgang-wallner@gmx.at

***Abstract*—Real-time systems rely on a distributed global time base. As any physical clock device suffers from noise, it is necessary to provide some kind of clock synchronization to establish such a global time base. Different clock synchronization methods have been invented for individual application domains. The Precision Time Protocol (PTP), which is specified in IEEE 1588, is another interesting option. It targets local networks, where it is acceptable to assume small amounts of hardware support, and promises sub-microsecond precision. PTP provides many different implementation and configuration options, and thus the Design Space Exploration (DSE) is challenging. In this paper we discuss the implementation of realistic clock noise and its synchronization via PTP in OMNeT++. The components presented in this paper are intended to assist engineers with the configuration of PTP networks.**

I. Introduction

Real-Time Systems (RTSs) are computer systems, where the progression of real time is of importance for the application which is carried out. Distributed RTSs are a special class of systems: here, several distributed computer systems fulfill a common task which is linked to the progression of real-time. For such an application, it is of utmost importance that each local node has reliable knowledge of the current time. A typical example for such a system could be several robots that work together in a production line: it's easy to image that the correct operation of the overall system relies on the timely coordination between the individual systems.

Any physical clock device suffers from imperfections[6], and an unsynchronized ensemble of clocks could drift away from each other without an upper bound over time. To construct a distributed global time base, different synchronization methods have been developed, each with unique benefits and drawbacks. The Precision Time Protocol (PTP), which is specified in IEEE 1588[1], presents another candidate to solve this problem. PTP promises sub-microsecond precision, while only requiring moderate costs. The target domain for PTP are local networks, where it is acceptable that each node provides hardware support for timestamping egressing and ingressing network frames.

PTP presents many different options for its implementation and configuration. To aid system designers with the task of DSE, a simulation framework for PTP is of great value. For my master thesis[2], I have developed such a simulation framework using OMNeT++[1]. This paper gives an overview over the OMNeT++ specific aspects of this project.

[1] https://omnetpp.org/

A. Related Work

This paper describes a subset of what I have been working of for my master thesis [2]. My thesis deals with the simulation of oscillator noise and the synchronization of oscillators via the PTP which is specified in IEEE 1588. An overall discussion of clocks and clock noise can be found in [6] and [7]. An algorithm on how to simulate a particular important noise type (Powerlaw Noise (PLN)) is given in [4]. My own noise simulation library is a modification for this approach to make it more suitable for Discrete Event Simulation (DES) environments like OMNeT++. In [3] an efficient implementation of the above mentioned noise simulation and its implementation in OMNeT++ are described, however to the best knowledge of the author this implementation is not freely available and the description contains not enough details to reproduce their results. Nevertheless this publication has served as an important guideline for the design of my own implementation.

The focus of this paper is on the OMNeT++-specific parts of my implementation. Another publication named *A Simulation Framework for IEEE 1588* will be presented at the 2016 International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication. This other publication will contain a more detailed discussion of my noise simulation library.

B. Focus of this work

The goal for my thesis was to implement a simulation of clock synchronization via PTP. As I had to keep the project focus limited due to time constraints, I could only implement a subset of all components that would be present in a real physical PTP network. What I have selected for my simulation is basically a minimal set of components to get a meaningful PTP simulation. It consists of the following items:

- **Noise generation:**
  - A generic clock model in OMNeT++, which can be extended by various noise implementations
  - A simulation library for Powerlaw Noise, which is a noise type commonly found in oscillators
- **Noise estimation:**
  - A PTP stack in OMNeT++, which implements most parts of the IEEE 1588-2008 standard, especially all clock and delay mechanism types
  - An adaption of this generic PTP stack to implement PTP over Ethernet

  - A generic model for PTP network nodes in OMNeT++
- **Noise cancellation:**
  - A basic clock servo in OMNeT++ based on a proportional-integral (PI) controller

All of the above mentioned components have been designed to be modular and extensible.

C. Structure of this work

In section II we present a short overview of how PTP works. Section III discusses clocks and clock noise, and how they are modeled in our simulation. The structure of our PTP implementation in OMNeT++ is described in section IV. Finally, section V sums up the current project state, while section VI gives an outlook to possible improvements.

II. Precision Time Protocol

This section intends to give a short overview over the principles of PTP. For a detailed discussion the interested reader is referred to the literature, e.g. [5].

A. Principal of operation

In a PTP network, the individual nodes will dynamically establishes a hierarchy with master-slave relationships. Two adjacent master and slave nodes then exchange information to allow the slave the estimation of its own clock offset relative to the master. Using this information, a slave node can then modify its own local clock device to minimize its offset.

B. Overview

The overall functionality of PTP consists of:

Establishment of a clock hierarchy PTP nodes exchange the attributes of their clock with their neighboring nodes. This information is then used by a distributed algorithm (the Best Master Clock (BMC) algorithm), to establish a loop free master-slave hierarchy on the network.

Distribution of time information PTP nodes with ports in the master state will periodically publish their current time value, using so called `Sync` messages.

Path delay estimation When a PTP node receives a `Sync` message on a slave port, it needs to estimate how long this message has been on its way. To fulfill this task, PTP specifies two methods for path delay estimation.

Configuration interface Information of PTP nodes is organized in standardized data sets. PTP also specifies a management interface for accessing these data sets.

C. Information exchange

An example for a possible communication between a PTP master and a PTP slave is shown in fig. 1. The master periodically sends information about its current local time to the slave via `Sync` messages. To estimate the time interval that `Sync` messages need until they arrive at the slave, the slave has to estimate the path delay. It does so by periodically sending `Delay_Requ` messages to the master, who then responds with `Delay_Resp` messages. The colored rectangles in the figure show the points in time when the frames are sent or received. The colored circles below the figure sketch the knowledge on the slave side. Using the collected information the slave is then able to estimate its own offset compared to the master.

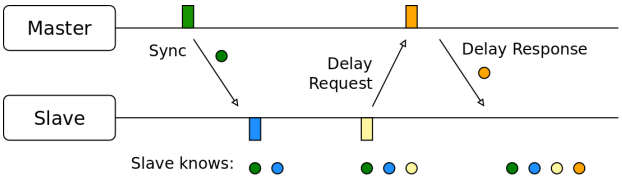


Figure 1: Synchronization principle of PTP.

III. Clock Model

In this section we discuss our approach for having realistic clock noise in our OMNeT++ simulation model.

A. Problem Statement

The systems we would like to simulate are distributed networks, where each network node has access to a local clock. We are interested in high precision clock synchronization, thus it is important for us that the individual clocks do not provide the exact value of real time, but only a local estimation.

The individual nodes rely on their clocks for two tasks:

- **Timestamping:** Reading the current value of the local clock to generate a timestamp for a local event.
- **Scheduling:** Registering an event together with a future timestamp at the local clock. The event will be delivered when the timestamp is reached on the local clock.

OMNeT++ provides Application Programming Interfaces (APIs) for these two tasks with respect to real-time (`simTime()` and `scheduleAt()`). We need to implement similar APIs for the simulation of (noisy) local clocks.

B. Clock Model

In [6] a model for a digital clock is given. This model consists of an oscillating device and a digital counter, which counts the oscillations. The simulation model for our clocks is based on this model. In a real clock, each of the two components could introduce noise. For our simulation, we are not interested in the exact location inside a clock where the noise is introduced, but only on the effect that can be observed on the outside. Thus, for our own model, we have modified the model from [6] as follows:

- Both the oscillator and the counter are assumed to be absolutely *perfect*
- In between these two we introduce a new component, called the noise generator. As the name already implies, its sole purpose is to introduce noise in our clock model.
- Finally, we have added a configurable scaling stage after the counter. This component can be used to synchronize the local clock to a reference clock using linear scaling.

A sketch of the complete clock model is given in fig. 2. Section III-D describes our implementation of this model.

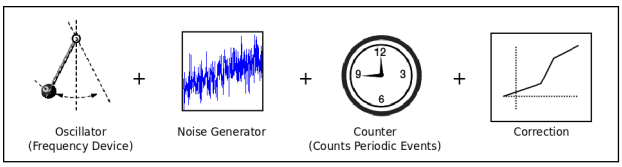


Figure 2: Clock model used in our simulation. The image is based on picture given in [6].

C. Clock Noise

Clock devices suffer from various influences that disturb their correct operation. These influences can be categorized as follows [6], [7]:

- Systematic influences (temperature dependence, aging, frequency drift)
- Stochastic noise

The stochastic influences in most frequency sources (including quartz oscillators) can be modeled as a combination of noise processes where the power spectral density is related to the frequency as $S_y(f) \propto f^{\alpha}$. This type of noise is referred to as Powerlaw Noise (PLN). For PLN found in common oscillators, it turns out that α has integer values in the interval $-2..2$.

While it would be desirable to have all a noise model for our simulations that covers all these effects, I could only implement a subset of them because of timely constraints. Depending on what kind of time intervals are most important for an application, different clock influences have to be considered or can be neglected. As we want to simulate high precision synchronization, I have considered it valuable to have an implementation of PLN, as it provides high-frequency noise as it is common in quartz oscillators. On the other hand, influences like aging can probable neglected without affecting the overall simulation results too much.

I have designed and implemented a library for the simulation of PLN, call *LibPLN*[2]. The actual PLN simulation is based on the approach by Kasdin and Walter [4], and ideas from [7] and [3].

D. Implementation

To reduce the complexity of our implementation, the individual clock tasks are implemented in different classes. Figure 3 shows the relationship between the individual components of our model.

a) Hardware Clock: The most basic class is the `HwClock`. Its purpose is the translation between the continuous and perfect real-time, and a discrete imperfect local time estimate. A hardware clock can not be adjusted, and its counter has no relationship to that of other clocks.

To fulfill its task, the hardware clock class relies on a component called `TdGen`, the Time Deviation (TD) generator. This component implements the current deviation of the local oscillator from the perfect real-time. It is this component that contains the noise model used in the simulation. In

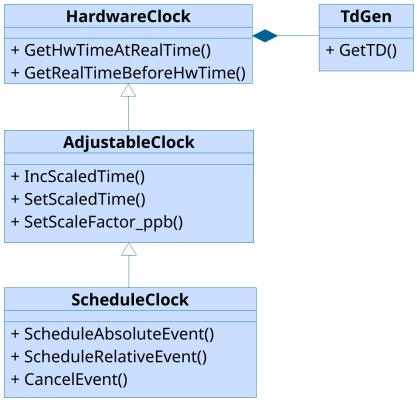


Figure 3: Hierarchy of our clock components

my implementation, it uses the above described *LibPLN* and only simulates Powerlaw Noise. Further s to the noise model could be implemented in this component (e.g. temperature dependence).

The local time estimate t_{est} for the real-time t can then be written as $t_{est} = t + TD(t)$. It is important that we can not assume to have an inverse for this function. But we can rely on the fact that the local time estimation is always increasing. This is important when we want to calculate the real-time when a specified local time estimate will be reached, as we need to this later to implement scheduling.

b) Adjustable Clock: Adjustable clocks provide an abstraction on top of hardware clocks. Their counter values can be modified, and their progress can be speed up or slowed down. These features enable us to synchronize multiple adjustable clocks to each other.

c) Schedule Clock: The finale clock class is the `ScheduleClock`, which provides an interface for other OMNeT++ components to schedule future events. Internally, it stores the scheduled events in an ordered list, and only the first one is actually scheduled. For the scheduling, it relies on the API provided by the hardware clock for real-time estimation together with OMNeT++'s `scheduleAt()` functionality.

IV. PTP Implementation

This section gives a brief overview over *LibPTP*, our OMNeT++-based implementation of PTP.

A. Node Architecture

PTP defines several different types of network nodes (Ordinary Clocks (OCs), Boundary Clocks (BCs) and Transparent Clocks (TCs)), as well as a large number of configuration options. In *LibPTP*, all these nodes are derived from a common base class, called `PTP_BasicNode`. The architecture of this base class is shown in fig. 4. Most components from LibPTP are based on standard models from the INET library[3].

The architecture shown here is a design choice with the intention to represent a basic PTP-capable network device. Any hardware related components have been placed on our

[2] https://github.com/w-wallner/libPLN

[3] https://inet.omnetpp.org

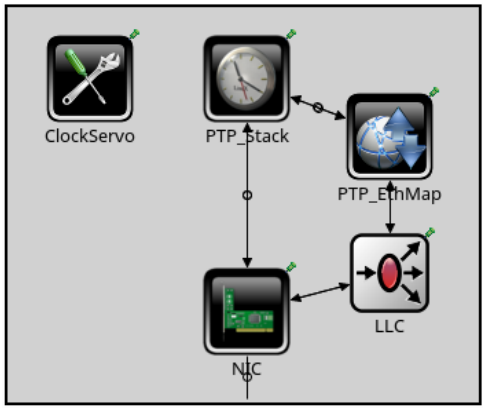


Figure 4: Architecture of a PTP Basic Node

simulated Network Interface Card (NIC). Components outside the NIC represent components that would typically be present as software in a real system. The node model here is loosely based on the architecture of a PC with a PTP-capable NIC. This is also why e.g. the clock that is used for the synchronization is shown as part of the simulated NIC.

The layout of our node model is one of the topics where community feedback for the improvement of *LibPTP* would be greatly appreciated.

a) PTP Stack: The most important component is the `PTP_Stack`, which is our implementation of the protocol as it is specified in IEEE 1588-2008 [1].

b) PTP Ethernet Mapping: PTP does not depend on a single lower layer technology, and various mappings are standardized, e.g for PTP over User Datagram Protocol (UDP), plain Ethernet, or different fieldbus systems. For our project, we focused on PTP over Ethernet, which is specified in Annex F of IEEE 1588-2008. The component `PTP_EthMap` implements this mapping.

c) Clock Servo: PTP only specifies a way to estimate the offset of a slave clock in reference to a master clock. How this offset is minimized is the responsibility of the individual implementations, in particular that of the implemented clock servo controller. Our simulation provides a generic interface for clock servos, and an example implementation using a PI-based clock servo design.

d) LLC: The component labeled `LLC` in fig. 4 is the Logical Link Control (LLC). It provides access to the NIC based on the EtherType field of Ethernet frames.

e) NIC: Each PTP node has a NIC with PTP hardware support. Figure 5 shows the internal architecture of our NIC model.

f) Clock: The clock inside the NIC implements the clock model described in section III.

g) Relay Unit: In case a network node with more than one physical network interface is simulated, it is the task of the `RelayUnit` component to distributed ingressing and egressing frames to the correct ports.

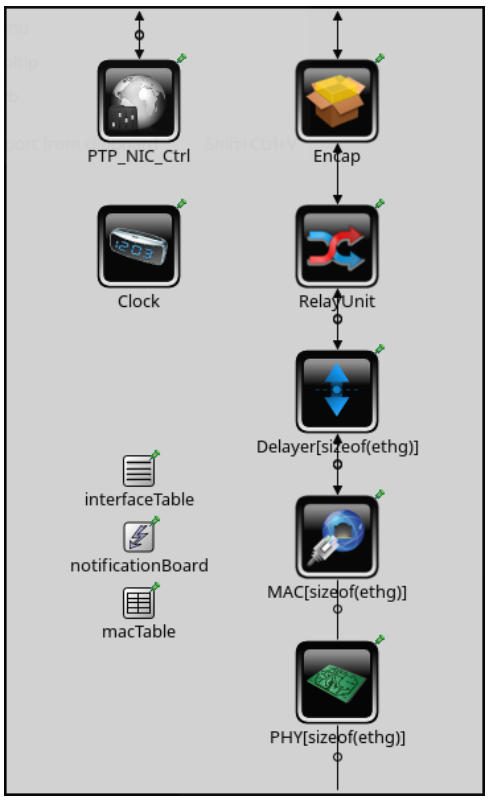


Figure 5: Internal architecture of our PTP-capable NIC

h) Delay: To simulate arbitrary scheduling/queuing delays, a special `Delayer` component is added to our simulation model.

i) MAC: The Media Access Control (MAC) is based on the `EtherMAC` from INET, but adds timestamping support for PTP frames.

j) PHY: The physical layer (PHY) model provides support for simulating asymmetric paths.

B. Simulations

Using the combination of *LibPTP* together with *LibPLN* allows us to carry out simulations of clock synchronization via PTP in OMNeT++.

a) Sync interval: On of the most important parameters for a PTP configuration is the interval for the sending of `Sync` messages. The synchronization interval in PTP is given via the `logSyncInterval` parameter as $T_{sync} = 2^{logSyncInterval}$, measured in seconds. Configuring a long interval might degrade the reachable clock synchronization. On the other hand, if an interval is configured which is too short, we might waste network bandwidth without any benefit for the precision of our global time-base. Using OMNeT++'s support for parameter studies, we can configure a PTP network and find out an optimal value empirically.

Suppose we have a simple network, consisting of only two PTP nodes. One of them has an excellent clock, and is the PTP master, while the other has a cheap oscillator and is the PTP slave. Carrying out a parameter study for possible

synchronization interval configurations could lead to a graph as it is shown in fig. 6. The image shows the worst-case jitter of the slave's time estimate relative to the time of the master.

The slave was simulated to have one of two oscillators: either a cheap quartz oscillator as it could be found in typical consumer electronic devices or a quartz typical for wrist watches. It can be seen that for both types of oscillators, the jitter decreases for shorter intervals, until it reaches an oscillator specific lower bound. Further decreasing the interval won't lead to better clock synchronization.

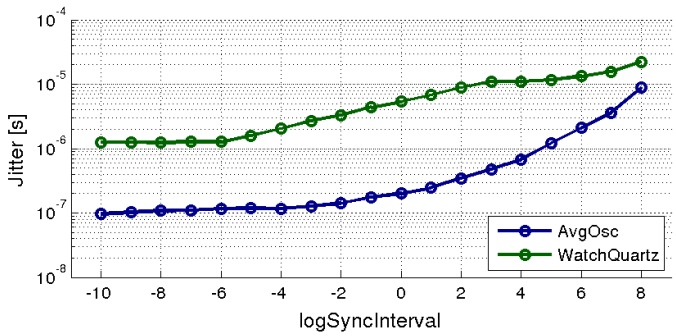


Figure 6: Parameter study for showing the effect of different synchronization intervals.

b) Tracing and debugging: *LibPTP* provides also sophisticated tracing and debugging support. All relevant PTP properties are available as signals and statistics. As an example, a trace of a PTP port state is shown in fig. 7.

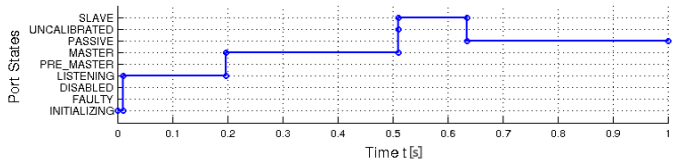


Figure 7: Trace of a PTP port state

Additionally, it is possible to enable various logging messages. For example, it is easily possible to get a complete trace output of every single decision that is carried out when the nodes execute the BMC or the Data Set Comparision (DSC) algorithms.

V. Conclusion

Our project has shown that it is feasible to implement realistic clock noise in OMNeT++, as well as to carry out simulations of clock synchronization via PTP. To best knowledge of the author, it is the most sophisticated implementation of IEEE 1588 in OMNeT++ which is freely available. All source code is released under open source licenses[4] on Github[5]. Additionally, also a portable library called *LibPLN* for the efficient simulation of Powerlaw Noise in DES-environment like OMNeT++ was implemented. Again, everything is available to the community as open source on Github[6]. The components presented here should help engineers with the difficult task of PTP DSE.

All of the implemented components have been designed to be extensible, so that other users can modify and improve them for their own projects.

As a side effect, having an OMNeT++-based implementation of PTP has proven to be an excellent tool when teaching other people how PTP works.

VI. Future Work

I have worked on this project mainly for my master thesis. During this time, I have worked on it in private. As my thesis is finished now, I have released all related source code under open source licenses, in the hope to find interested community members for further enhancements.

Future improvements to *LibPTP* could include:

- **Extending PTP options:** Currently only a subset of the standardized PTP options is implemented in *LibPTP*. Extending this support would further enhance the capabilities for DSE. Of interest would be e.g. unicast support, PTP over UDP or support for different PTP profiles.
- **Mainlining LibPTP:** *LibPTP* is derived from components in the INET library, but it is not related to upstream development. This implies that the projects will drift further apart. It would be of great benefit to try to merge at least some of the work upstream.
- **Improving the noise model:** Currently, only stochastic noise of oscillators is modeled (using *LibPLN*). It would be interesting to add support for deterministic influences like temperature to the simulation model. This would allow the simulation to provide even more realistic results.

VII. Acknowledgements

I would like to thank my supervisor Armin Wasicek for his support during my master thesis. I am grateful to both the OMNeT++ and INET communities: without the availability of their excellent tools/libraries, my PTP implementation would not have been possible.

References

[1] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems* IEEE Std. 1588-2008, 2008.

[2] W. Wallner, *Simulation of Time-synchronized Networks using IEEE 1588-2008* Master's thesis, Faculty of Informatics, Vienna University of Technology, 2016.

[3] G. Gaderer et al, *Achieving a Realistic Notion of Time in Discrete Event Simulation* International Journal of Distributed Sensor Networks, 2011.

[4] N. Kasdin and T. Walter, *Discrete Simulation of Power Law noise* Proceedings of the 1992 IEEE Frequency Control Symposium, 1992.

[5] J. C. Eidson, *Measurement, Control, and Communication Using IEEE 1588* Springer, 2006.

[6] D. W. Allan, N. Ashby, C. C. Hodge, *Application Note 1289: The Science of Timekeeping* Agilent Technologies, 2000.

[7] W. J. Riley, *NIST Special Publication 1065: Handbook of Frequency Stability Analysis* National Institute of Standards and Technology, U.S. Department of Commerce, 2008.

[4] Most parts are released under the GPL, the rest under the BSD license.

[5] https://github.com/w-wallner/libPTP

[6] https://github.com/w-wallner/libPTL

Stacked-VLAN-Based Modeling of Hybrid ISP Traffic Control Schemes and Service Plans Exploiting Excess Bandwidth in Shared Access Networks

Kyeong Soo Kim
Department of Electrical and Electronic Engineering
Xi'an Jiaotong-Liverpool University
Suzhou, 215123, P. R. China
Email: Kyeongsoo.Kim@xjtlu.edu.cn

***Abstract*—The current practice of shaping subscriber traffic using a token bucket filter by Internet service providers may result in a severe waste of network resources in shared access networks; except for a short period of time proportional to the size of a token bucket, it cannot allocate excess bandwidth among active subscribers even when there are only a few active subscribers. To better utilize the network resources in shared access networks, therefore, we recently proposed and analyzed the performance of access traffic control schemes, which can allocate excess bandwidth among active subscribers proportional to their token generation rates. Also, to exploit the excess bandwidth allocation enabled by the proposed traffic control schemes, we have been studying flexible yet practical service plans under a hybrid traffic control architecture, which are attractive to both an Internet service provider and its subscribers in terms of revenue and quality of service. In this paper we report the current status of our modeling of the hybrid traffic control schemes and service plans with OMNeT++/INET-HNRL based on IEEE standard 802.1Q stacked VLANs.**

***Index Terms*—ISP traffic control, excess bandwidth allocation, stacked VLANs.**

I. Introduction

The resource sharing in shared access networks — like cable Internet based on hybrid fiber-coaxial (HFC) networks or passive optical networks (PONs) — is a key to achieving lower infrastructure cost and higher energy efficiency. The full sharing of the bandwidth available among subscribers in a shared access network, however, is hindered by the current practice of traffic control by Internet service providers (ISPs), which is illustrated in Fig. 1; due to the arrangement of traffic shapers (i.e., token bucket filters (TBFs)) and a scheduler in the access switch, the capability of allocating available bandwidth by the scheduler is limited to the *traffic already shaped* per service contracts with subscribers [1], [2].

Even though the allocation of excess bandwidth in a shared link has been discussed in the general context of quality of service (QoS) control (e.g., [4]), it is recently when the issue was studied in the specific context of ISP traffic control in shared access [5], [6]. Based on the ISP traffic control schemes proposed in [5] and [6], we have been studying the design of flexible yet practical ISP service plans exploiting the excess bandwidth allocation in shared access networks under a hybrid ISP traffic control architecture in order to gradually introduce the excess bandwidth allocation while providing backward

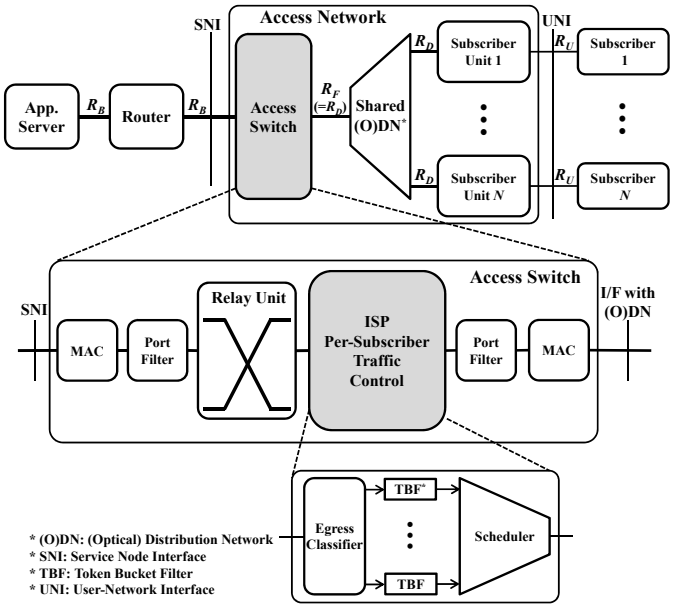


Fig. 1. Overview of current practice of ISP traffic control in shared access (shown for downstream traffic only) [3].

compatibility with the existing traffic control infrastructure [3]. To the best of our knowledge, our work in [3], [5], [6] is the first effort to study the issue of enabling excess bandwidth allocation among the subscribers, together with its business aspect, in the context of ISP traffic control in shared access.

In this paper, we report the current status of our modeling of the hybrid ISP traffic control schemes and service plans exploiting excess bandwidth in shared access networks with OMNeT++ [7] and INET-HNRL[1] based on the stacked virtual local area networks (VLANs) of IEEE standard 802.1Q [9].

II. Review of Hybrid ISP Traffic Control for Shared Access

In this section, we briefly review the hybrid ISP traffic control schemes and service plans for shared access that we

[1]A fork of INET framework (rev. INET-20111118) [8] and available at http://github.com/kyeongsoo/inet-hnrl, which requires OMNeT++ version 4.6 and later.

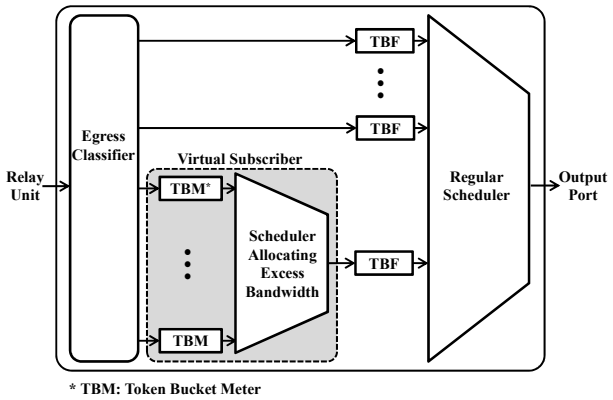


Fig. 2. Hybrid ISP traffic control for a flexible service plan exploiting excess bandwidth allocation [3].

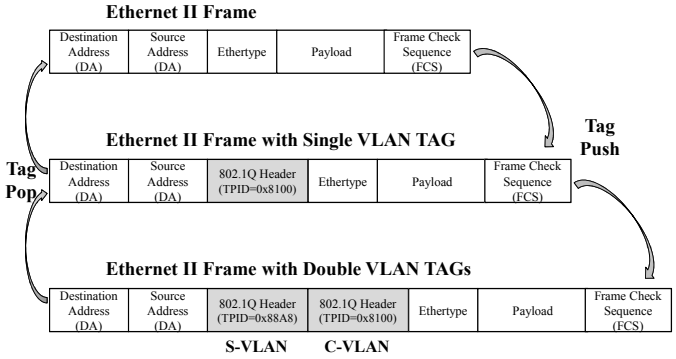


Fig. 3. Frame formats for VLAN stacking.

proposed in [3].

Fig. 2 shows the proposed architecture for hybrid ISP traffic control, where there coexist subscribers for the current flat-rate service plan and those for a new service plan fully sharing the bandwidth among them. For backward compatibility with the existing traffic control and pricing schemes, the new service plan subscribers are grouped together and treated as one *virtual* subscriber under the flat-rate service plan; at the same time, the traffic from each subscriber of the new service plan is individually controlled by an ISP traffic control scheme enabling excess bandwidth allocation within the group. The migration toward fully-shared access will be completed when all the subscribers of the flat-rate service plan move to the new service plan exploiting excess bandwidth allocation.

Note that, for the new service plan to be acceptable, it is desirable that there should be no disadvantage in adopting the new service plan for both ISP and its subscribers compared to the existing flat-rate service plan. In this regard, we can derive requirements for the new service plan to meet in terms of parameters for existing flat-rate service plans, including monthly price, token generation rate, and token bucket size. Interested readers are referred to [3] for details.

III. Modeling of Hybrid ISP Traffic Control Schemes and Service Plans Based on Stacked-VLANs

As discussed in [10], we have already implemented models of the shared access network shown in Fig. 1 based on VLAN as part of INET-HNRL, because we want abstract models that can provide features common to specific systems (e.g., cable Internet and Ethernet PON (EPON)), while being practical enough to be compatible with other components and systems of the whole network. In the VLAN-based shared access models, we use a VLAN identifier (VID) to identify each subscriber, which is similar to the service identifier (SID) in cable Internet and the logical link identifier (LLID) in EPON.

For the implementation of models for the hybrid ISP traffic control shown in Fig. 2, we can think of two distinct approaches, i.e., an integrated approach where we implement the whole scheduling as one system (e.g., based on the hierarchical token bucket (HTB) scheduler [4]) and a modular approach where we integrate separate schedulers (e.g., a scheduler based on TBF shaping and a DRR-based scheduler enabling excess bandwidth allocation [6]) into one. Considering the ease of the management of two separate groups of subscribers and the upgradability of the component scheduler allocating excess bandwidth independently of the traditional one based on TBFs, we have chosen a modular approach and again based our implementation on VLAN.

Unlike existing models based on a single VLAN tag per frame, we need two different ways of identifying frames from the subscribers for the new hybrid traffic control scheme and service plan: As for the existing TBF-based traffic control scheme, the whole frames from those subscribers need to be identified and treated as a group (i.e., one virtual subscriber) for traffic shaping and scheduling; as for the new excess-bandwidth-allocating traffic control scheme, on the other hand, the frames from each subscriber need to be identified and treated as a separate flow. Fortunately, this requirement of hierarchical identification of Ethernet frames under the new hybrid traffic control scheme can be met by the technique of *stacked VLANs* (also called *provider bridging* and *Q-in-Q*), which is now part of IEEE standard 802.1Q [9]. The change of Ethernet frame formats related with the VLAN stacking and two tag operations are shown in Fig. 3. Note that the tag protocol identifier (TPID) of the second service VLAN (S-VLAN) tag is set to a value of 0x88A8, different from the value of 0x8100 for the first customer VLAN (C-VLAN) tag.

Fig. 4 shows stacked-VLAN-based modeling of a shared access network with hybrid ISP traffic control, while Figs. 5 and 6 show the Ethernet switch module for ONUs, OLTs, and access switches, and the Ethernet MAC module implementing hybrid ISP traffic control, respectively; as for the traffic control schemes enabling excess bandwidth allocation, there are implemented two queue types, i.e., *CSFQVLANQueue5* for the algorithm based on core-stateless fair queueing (CSFQ) [5] and *DRRVLANQueue3* for the algorithm based on deficit round-robin (DRR) [6].

First, the "olt_c" access switch carries out individual traffic control based on the customer VID (C-VID) of a frame with a single C-VLAN tag, which is assigned to each subscriber, and sends resulting frames to the second access switch node

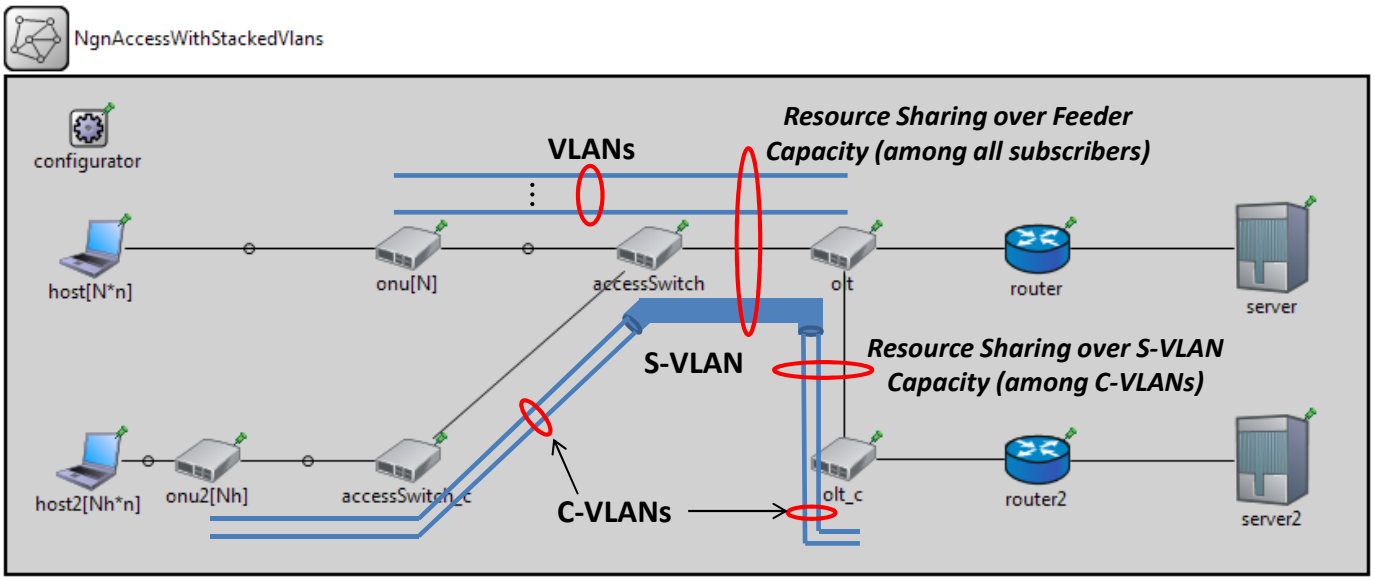


Fig. 4. Stacked-VLAN-based modeling of an access network with hybrid ISP traffic control.

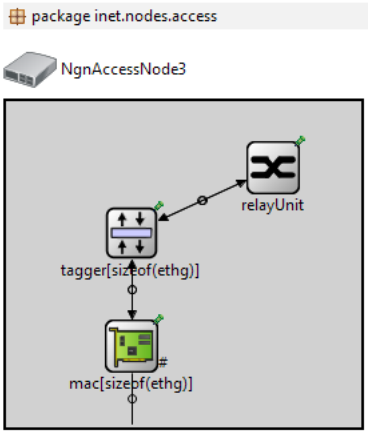


Fig. 5. Ethernet switch module (*NgnAccessNode3*) with stacked-VLAN capabilities (for ONUs, OLTs, and access switches shown in Fig. 4).

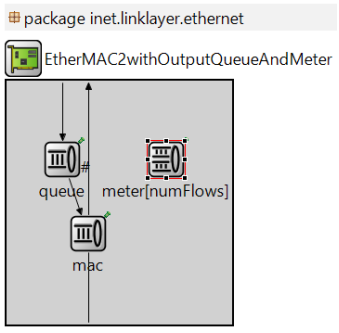


Fig. 6. Ethernet MAC (*EtherMAC2*) module with a queue and a traffic meter for modeling of hybrid ISP traffic control.

"olt". At the "olt", the C-VLAN frames are grouped together with the second S-VLAN tag (i.e., VLAN stacking) and go through another traffic control together with frames from other subscribers with normal (i.e., unstacked) VLAN tags. In this way, traffic for the subscribers of the new hybrid traffic control scheme and service plan goes through two stages of traffic control, i.e., one at the "olt_c" exploiting excess bandwidth allocation and the other at the "olt" based on traditional TBF-based traffic shaping.

In implementing models of the hybrid traffic control in shared access based on stacked VLANs, we tried to meet the following major requirements:

- *Backward compatibility* with the existing VLAN implementations in INET-HNRL, including
  - *EthernetFrameWithVLAN* message format
  - *MACRelayUnitNPWithVLAN* and *VLANTagger* modules
- *Expandability* to stack more than two VLAN tags

Consider the original definition of *EthernetFrameWithVLAN* message shown in Fig. 7 (a). Because the *MACRelayUnitNPWithVLAN* switching module is based on the *vid* field of the *EthernetFrameWithVLAN* message, which is directly accessible by the *getVid()* member function, we had to keep these fields in the new definition of *EthernetFrameWithVLAN* message. For stacking of VLAN tags, on the other hand, we need to introduce *innerTags* field based on the *stack* C++ type, which is shown in Fig. 7 (b) and ignored by the existing modules based on the original definition of *EthernetFrameWithVLAN* message, including *MACRelayUnitNPWithVLAN* module. In this way, we can meet both the requirements.

Note that in the current implementation of stacked VLANs, broadcasting is not allowed across the hierarchies of stacked VLANs. In the shared access network model shown in Fig. 4,

```
packet EthernetIIFrameWithVLAN extends EthernetIIFrame
{
    uint16_t tpid = 0x8100; // tag protocol identifier (16 bits; set to 0x8100)
    uint8_t pcp; // priority code point for IEEE 802.1p class of service (3 bits; 0 (lowest) to 7 (highest))
    bool dei; // drop eligible indicator (1 bit)
    uint16_t vid; // VLAN identifier (12 bits; 0x000 and 0xFFF are reserved, which allows up to 4094 VLANs)
}
```

(a)

```
cplusplus {{
#include <stack>
#include "VLAN.h" // define VLANTag struct
typedef std::stack<VLANTag> VLANTagStack;
}}

class noncobject VLANTagStack;

packet EthernetIIFrameWithVLAN extends EthernetIIFrame
{
    uint16_t tpid; // tag protocol identifier (16 bits; set to 0x8100 for C-TAG & 0x88A8 for S-TAG)
    uint8_t pcp; // priority code point for IEEE 802.1p class of service (3 bits; 0 (lowest) to 7 (highest))
    bool dei; // drop eligible indicator (1 bit)
    uint16_t vid; // VLAN identifier (12 bits; 0x000 and 0xFFF are reserved, which allows up to 4094 VLANs)

    // optional; IEEE 802.1Q-in-Q stacked VLANs.
    VLANTagStack innerTags; // based on std::stack
}
```

(b)

Fig. 7. Message definitions of Ethernet II frame with VLAN support: (a) Without and (b) with VLAN stacking.

for example, broadcasting is possible among normal VLANs or C-VLANs within the same S-VLAN. Broadcasting over the hierarchies of stacked VLANs requires the modification of the learning mechanism implemented in the current *MACRelayUnitNPWithVLAN* module.

IV. Summary

In this paper we discuss the issues in current practice of ISP traffic shaping and related flat-rate service plans in shared access networks and review alternative service plans based on new hybrid ISP traffic control schemes exploiting excess bandwidth. We also report the current status of our modeling of the hybrid ISP traffic control schemes and service plans with OMNeT++/INET-HNRL based on stacked VLANs.

In implementing models of the hybrid traffic control in shared access based on stacked VLANs, we maintain backward compatibility with the existing modules for Ethernet switching and VLAN tagging and yet enable the support of stacking of multiple VLAN tags by clever modification of the message definition for Ethernet frame with VLAN tags.

Acknowledgment

This work was supported by Xi'an Jiaotong-Liverpool University Research Development Fund (RDF) under grant reference number RDF-14-01-25.

References

[1] S. Bauer, D. Clark, and W. Lehr, "PowerBoost," in *Proc. HomeNets'11*. New York, NY, USA: ACM, Aug. 2011, pp. 7–12.

[2] L. Farmer and K. S. Kim, "Cooperative ISP traffic shaping schemes in broadband shared access networks," in *Proc. the 4th International Workshop on Fiber Optics in Access Network (FOAN 2013)*, Sep. 2013, pp. 21–25.

[3] K. S. Kim, "Toward fully-shared access: Designing ISP service plans leveraging excess bandwidth allocation," in *Proc. ICTC 2014*, Busan, Korea, Oct. 2014, pp. 897–900.

[4] M. Devera. Linux HTB home page. [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/

[5] K. S. Kim, "On the excess bandwidth allocation in ISP traffic control for shared access networks," *IEEE Commun. Lett.*, vol. 18, no. 4, pp. 692–695, Apr. 2014.

[6] ——, "On guaranteeing the quality of service of conformant traffic in excess bandwidth allocation for shared access networks," in *Proc. IEEE Sarnoff Symposium 2015*, Aug. 2015.

[7] A. Varga, "The OMNeT++ discrete event simulation system," in *Proc. the European Simulation Multiconference (PESM2001)*, Prague, Czech Republic, Jun. 2001, pp. 319–324. [Online]. Available: http://www.omnetpp.org/

[8] A. Varga *et al.* INET framework for OMNeT++ 4.0. [Online]. Available: http://inet.omnetpp.org/

[9] IEEE Computer Society, *IEEE Std 802.1Q™-2014, IEEE Standard for local and metropolitan area networks: Bridges and bridged networks*, IEEE Std., Nov. 2014.

[10] K. S. Kim, "The effect of ISP traffic shaping on user-perceived performance in broadband shared access networks," *Computer Networks*, vol. 70, pp. 192–209, Sep. 2014.

An OMNeT++ based Framework for Mobility-aware Routing in Mobile Robotic Networks

Benjamin Sliwa, Christoph Ide and Christian Wietfeld
Communication Networks Institute
TU Dortmund University
44227 Dortmund, Germany
e-mail: {Benjamin.Sliwa, Christoph.Ide, Christian.Wietfeld}@tu-dortmund.de

***Abstract*—In this paper, we propose a cross-layer extension for the INETMANET framework of OMNeT++, which utilizes mobility control knowledge to enhance the forwarding of routing messages. The well-known mobility meta-model from Reynolds is used to provide a realistic representation of the mobility behavior of autonomous agents with respect to the various influences those agents have to face in real-world applications. Knowledge about the current mobility properties and its predicted development is used by mobility-aware routing mechanisms in order to optimize routing decisions and avoid path losses at runtime. In a proof of concept evaluation we show that our proposed methods can achieve significant improvements to the robustness of communication paths.**

I. Introduction

Objective Modular Network Testbed in C++ (OMNeT++) [1] is a well-established network simulation framework and has been extended with many mobile ad-hoc network (MANET) routing protocols by the INETMANET framework. Although mobile nodes are supported and several generic mobility models are available, geo-assisted routing protocols are not yet part of the framework. Furthermore, the provided mobility models are not capable of providing a realistic represention of the high number of challenges mobile robotic systems have to face in real-world applications. Besides exploration tasks, collision avoidance and maintenance of the swarm coherence are further important aspects, which influence the mobility behavior. In our recent work [2], we presented the mobility-aware routing protocol B.A.T.Mobile and its trajectory prediction method, which leverages application layer mobility control knowledge for precise position estimations. We demonstrated its capability to provide reliable communication even under highly challenging channel conditions and showed it is significantly outperforming the established protocols in highly dynamic networks. In this paper, we propose an extension of the capabilities of OMNeT++/INETMANET. Our goal is to provide a realistic mobility model for autonomous agents and to offer a solid basis for the design of further cross-layer and mobility-aware routing protocols. Therefore, we abstract the principles used by B.A.T.Mobile and propose a base module for protocols, which utilize knowledge about the controlled mobility trajectories to optimize the routing decisions. The remainder of this paper is structured as follows: After discussing the related work, we present the system model of our framework and discuss the relevant submodules. In the next section, we describe the simulation setup for a defined reference scenario. Finally, an impression about the potentials of the proposed approach is given by presenting results for a proof of concept evaluation.

II. Related Work

In the field of MANETs various protocols have been proposed to face the various challenges mobile nodes encounter in real-world applications. Fig. 1 illustrates the different categories of routing approaches and names example protocols. *Topology-based* protocols use knowledge about the links between nodes for their decision-making and form the most popular category of MANET routing approaches. Further distinction is done into proactive [3][4], reactive [5][6] and hybrid [7][8] protocols. Recently, the *geo-based* approach [9][10] has received a lot of attention in the scientific community. Nodes determine their own position using a global navigation satellite system (GNSS) receiver and manage knowledge about the positions of the other network participants with a *location service*. The selection of forwarder nodes is performed depending on their distance to the destination. Different forwarding schemes are compared in [11]. Since link-state information is not considered by most geo-based protocols, additional recovery strategies are needed if the location-based topology map mismatches the actual channel conditions. In this paper, we combine the *topology-based* and the *geo-based* approach to take advantage of both paradigms. The *mobility-aware* routing class uses link-state information as well as knowledge about the mobility of the nodes to enhance the routing process. In

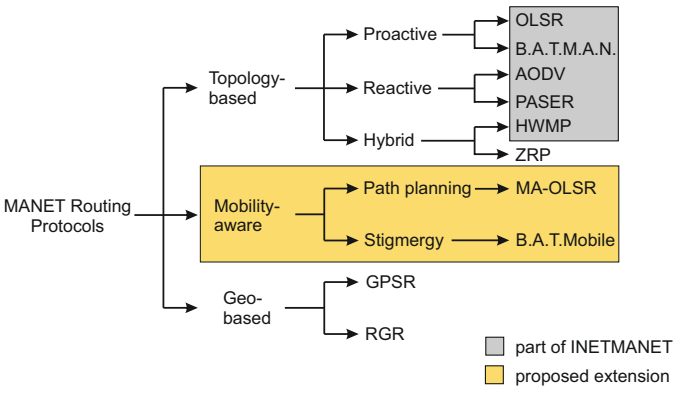


Fig. 1. Classification of MANET routing protocols

earlier work we presented the B.A.T.Mobile protocol, which uses stigmergic principles. Later in this paper, we will describe a mobility-aware path planning algorithm and give a basic evaluation of the method as an extension to OLSR, called mobility-aware OLSR (MA-OLSR).

III. Design and Implementation of the Framework

The proposed cross-layer system model is illustrated in Fig. 2. The modules' implementations are derived by inheriting from the base modules of the INETMANET framework. The novel *MobilityAwareHost.ned* acts as a compound module for all logical submodules required for mobility-aware routing.

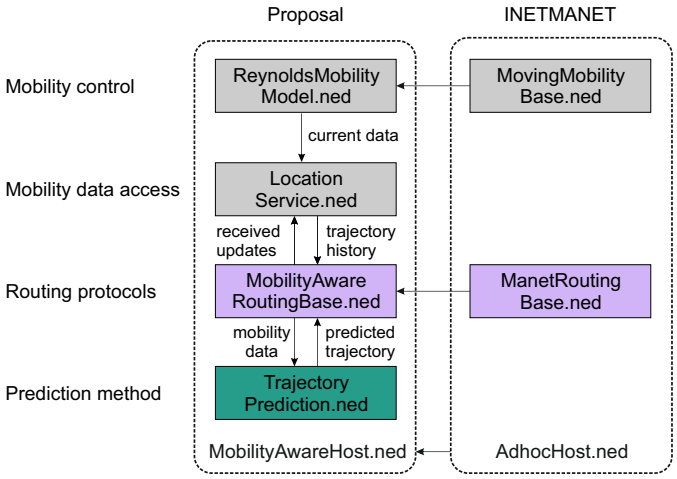


Fig. 2. Cross-layer approach: utilization of mobility control layer information to enhance the routing process

In the following sub-chapters, we give a detailled description of the implementation of the mobility meta-model as well as the routing principles.

A. Implementation of a realistic mobility meta-model for autonomous agents

In the field of MANETs, mobile hosts are often modelled using established generic mobility models. While this approach is sufficient for protocol evaluations of independent nodes, it is not capable of providing a realistic representation of the mobility behavior of cooperative autonomous agents. In real-world applications, mobile robots are influenced by the mobility characteristics of other nodes in order to avoid collisions, keep up communication paths and to fulfill cooperative tasks (e.g. exploration and network provisioning). Reynolds has proposed a basic mobility model for autonomous agents in [12] that has received great acceptance inside the scientific community. We use this well-known approach as a general logical meta-model for our mobility modules. One of its key advantages is the intrinsic support for combining different behavior models, which can be used to increase the capabilities of situation-aware reacting. The model consists of three layers, which are illustrated in Fig. 3.

- *Action Selection* defines the global tasks the mission has to fulfill. Examples are exploration and network provisioning. Different agent classes (e.g., scouts and relays) can be defined in order to pay attention to specific requirements of these tasks.
- *Steering* determines how a specific task is handled. Usually multiple specific steerings are handled simultaneously to take the various challenges into account.
- *Locomotion* describes the physical movement of the nodes. For simulations a model of the vehicles' movement characteristics is used, while real-life vehicles control the actual motors.

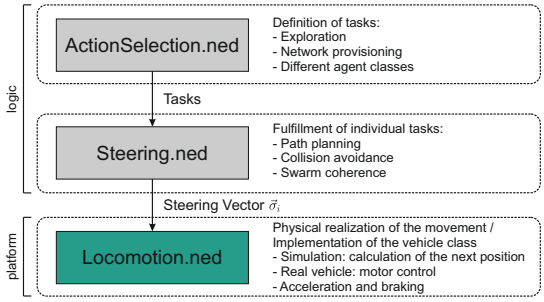


Fig. 3. The Reynolds mobility model provides a meta-model for the integration of mobility algorithms and consists of multiple layers

The management of the interaction between the layers is performed by the *ReynoldsMobilityModel.ned* module. It inherits from the *MovingMobilityBase.ned* module and uses the parent module's update timer. With each mobility update all steerings are triggered and the steering vector $\vec{\sigma}_i$ is computed as their weighted mean. Different weights are applied to specify the importance of individual steerings, e.g., collision avoidance might be of higher priority than exploration if the distance between vehicles is low. The result vector is then passed to the *Locomotion.ned* module, which maps the desired movement to a travelled path with respect to the physical capabilities of the vehicle. If different vehicle types (e.g., multicopters and planes) are modelled using dedicated locomotion classes, they can use the same steerings, increasing the portability of mobility algorithms. Many application scenarios for mobile robotic networks (e.g., exploration of hazardous areas) focus on cooperative task fullfilment instead of individual mobility. In [13], Reynolds describes three basic rules for the behavior of individual agents when travelling in swarms, which are illustrated in Fig. 4.

- *Separation* is used to keep up a minimal distance between individual agents. In the field of Unmanned Aerial Vehicles (UAVs) this aspect represents the *collision avoidance* mechanism. Recent implementations often make use of potential fields [14], where obstacles and other agents cause repelling forces.
- *Cohesion* is the antagonist of separation and describes the maintenance of the swarm structure, which has a high impact on the number of available communication links. This aspect is important if network provisioning is the main task and it is often represented using potential fields with attracting forces to the swarm centroid [15].
- *Alignment* describes the focus on a swarm target and addresses the fulfillment of cooperative tasks such as plume exploration [16].

Decent models of cooperative swarms have to address all of these aspects. Individual behavior algorithms can be implemented as steerings, which are then triggered by the meta-model. Knowledge about the positions of other swarm participants is one of the key requirements for the design of cooperative mobility algorithms. This information is managed individually by each agent using the *LocationService.ned* module, which acts as a mediator between the mobility control and the routing layer. After each position change, the own position entry in the location service is refreshed. Mobility information contained in routing messages is used to update the current knowledge about the sender node in the location service proving a solid data base for path planning. The data entries contain different types of individual mobility information.

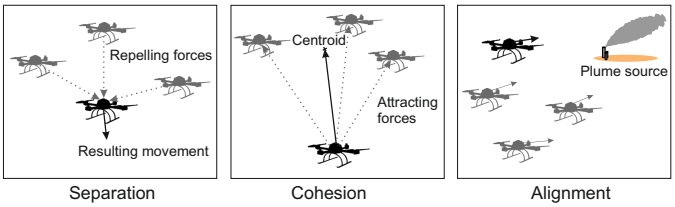


Fig. 4. Reynolds' three basic rules for the behavior of individual agents travelling in swarms

- The *position* $\vec{P}_i$ represents the measured position value in the mobility update iteration i and it is derived from the real position used by the *MovingMobilityBase.ned* module. In order to take the influences of real-world GNSS receiver inaccuracies into account it can be distorted with a noise vector for a defined maximum positioning error. Further extensions could contain different positioning methods like using the characteristics of wireless signals for location estimations.
- The *steering vector* $\vec{\sigma}_i$ is the weighted superposition of individual steering results and represents the current movement vector to the desired position in the next iteration $i + 1$.
- *Waypoints* represent the planned trajectory of the agent. One example for obtaining this information in future real-world applications is the navigation system of autonomous cars.

Apart from modelling mobility behavior as steerings, which are executed in OMNeT++, mobility data can also be loaded from trace files. With this approach mobility data obtained from real-world measurements as well as from other simulators can directly be used in order to analyze the routing characteristics of the network. With the availability of mobility data managed by the location service, nodes are able to predict future positions for a defined prediction width N_p with the help of the *TrajectoryPrediction.ned* module. This information can be used by the routing layer in order to select the path with the highest assumed stability to avoid packet losses. N_p is a multiple of the mobility update interval Δt_u and represents the tradeoff between the influence of the current and the future link-state for the total link quality assessment.

B. Mobility-aware routing

The selection of forwarder nodes for a given destination is the main task of the routing process and therefore a key factor for providing reliable communication. The mobility-aware approach uses current mobility data as well as predicted knowledge to improve the robustness of communication paths. There are two general classes for using predicted mobility data for enhancing the routing decisions. *Predictive path planning* requires all nodes to propagate their current mobility information through the network. Every node maintains a database of the mobility information of all others nodes, which is used to predict the future state of the whole network. The prediction is done with the trajectory prediction algorithm of [2]. Alg. 1 describes the selection of the best forwarder node for a given destination. It assumes the routing protocol maintains topology information in a link map. Future links are assumed to be valid if their predicted distance d is below a maximum distance d_{max}, which is obtained from a channel model for a defined receiver sensitivity $P_{e,min}$. This channel model represents the assumption of the channel conditions from the agent's view and it is not equal to the one used by OMNeT++ to calculate the path loss. The actual distribution of mobility information is performed with the *MobilityUpdatePacket* (see Fig. 5) and can either be broadcasted or piggybacked to existing routing messages. If mobility update packets are lost, the routing process may use outdated information.

Algorithm 1 Predictive Geo-based Dijkstra

```
1:  function FINDBESTNEIGHBOR(destination, linkMap, channelModel, dijkstra, P_e,min)
2:      bestNeighbor ← null
3:      links ← {}
4:      d_max ← channelModel.calculateDistance(P_e,min)
5:      for link in linkMap do                    ▷ link is currently available
6:          node_1 ← link.firstNode
7:          node_2 ← link.secondNode
8:          d ← calculatePredictedDistance(node_1, node_2)
9:          if d < d_max then                     ▷ link is available in the future
10:             links.append(link)
11:         end if
12:     end for
13:     path ← dijkstra.findPath(destination, links)
14:     if path.length > 0 then
15:         bestNeighbor ← path.first
16:     end if
17:     return bestNeighbor
18: end function
```

With the *stigmergic* approach used by B.A.T.Mobile, all agents periodically flood *PathScorePackets* through the network, which are updated by each forwarder with the current and the predicted forwarder position and a score S_i about the path quality of the reverse path to the sender. The value for S_i is calculated with the *path score metric*, which takes the absolute distance and the predicted distance development to the forwarder into account. Future implementations of routing protocols can inherit from the abstract *MobilityAwareRoutingBase.ned* module, which provides access to current and predicted mobility data as well as the proposed routing metrics.

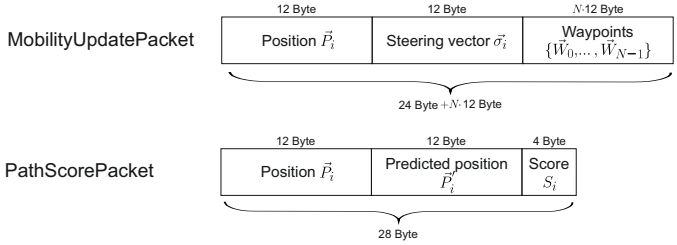


Fig. 5. Message types for distributing the required mobility information through the network. The *MobilityUpdatePacket* contains the individual mobility information needed for predictive path planning, while the *PathScorePacket* only contains the prediction results needed for the stigmergic approach.

IV. Simulation-based system model

To provide an evaluation scenario, we consider a swarm of autonomous agents performing an exploration task in a defined mission area. For our reference scenario, we evaluate the air-to-ground packet delivery ratio (PDR) of a videostream, which is transmitted from a randomly selected agent to a static base station as User Datagram Protocol (UDP) constant bitrate (CBR) data. Fig 6 illustrates the scenario. A *Controlled waypoint* mobility model is used for the exploration task in order to provoke frequent changes of the network topology during runtime. The waypoints are determined randomly but in contrast to the well-known *Random waypoint* model, all nodes know about their future waypoints, making this information utilizable for the routing process. Additionally a collision avoidance mechanism is used and triggered if the distance of two nodes is below a defined threshold in order to avoid dangerous situations. Both mobility algorithms are implemented as *steerings* and handled by the meta-modell. The influence

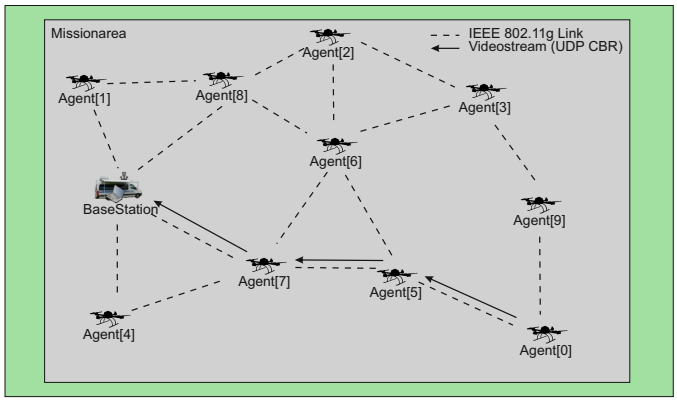


Fig. 6. Simulation of the reference scenario in OMNeT++

of packet losses is evaluated considering two different channel models. We use a *Friis* channel model for rural environments and a *Nakagami* (m=2) model for urban scenarios in order to identify implications of different packet loss probabilities for the routing mechanisms. The mobility-aware protocols MA-OLSR and B.A.T.Mobile are compared to their base versions OLSR and B.A.T.M.A.N. The goal of the evaluation is a comparison of the approaches predictive path planning and stigmergy in order to identify the one which is able to benefit most from the integration of mobility information and is a promising candidate for further extensions and evaluations. Tab. I shows the reference parameter assignment.

TABLE I
Simulation Parameters for the Reference Scenario in OMNeT++/INETMANET

Simulation parameter	**Value**
Mission area	500 m x 500 m x 250 m
Number of agents	10
Steering[0] (Exploration)	Controlled Waypoint
Steering[0].weight	1
Steering[1] (Collision Avoidance)	LocationBasedCollAvoidance
Steering[1].weight	10
Steering[1].minDistance	30 m
Locomotion	UAVLocomotion
Mobility update interval Δt_u	250 ms
Movement speed	50 km/h
Channel model	Friis / Nakagami ($\alpha = 2.75$)
Videostream bitrate	2 Mbit/s
Videostream packet size	1460 Byte
Telemetry broadcast interval	250 ms
Telemetry packet size	1000 Byte
OGM broadcast interval (B.A.T.M.A.N.)	0.5 s
HELLO interval (OLSR)	0.5 s
Topology Control (TC) interval (OLSR)	1 s
Medium Access Control (MAC) layer	IEEE802.11g
Bitrate	54 Mbit/s
Transport layer protocol	UDP
Transmission power	100 mW
Carrier frequency	2.4 GHz
Receiver sensitivity ($P_{e,min}$)	-83 dBm
Simulation time per run	300 s
Number of simulation runs	50
Mobility data history size N_h	5
Prediction width N_p	15

V. Proof of concept evaluation

In this section, we present example results obtained with the extension to INETMANET in order to prove its capability of providing a base for implementing further mobility-aware routing protocols and to show the efficiency of the described principles. The results show the 0.95 confidence interval of 50 simulation runs. Fig. 7 compares the PDR values for OLSR and MA-OLSR, which uses the presented *predictive*

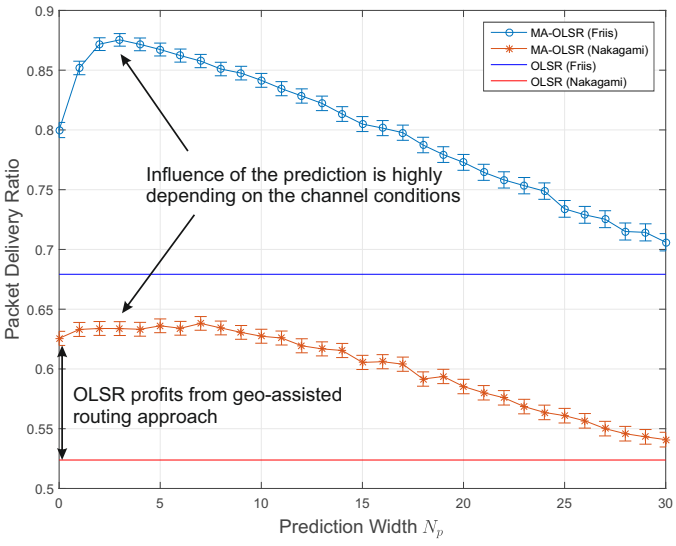


Fig. 7. Performance evaluation of OLSR using predictive path planning

path planning algorithm in two different channel models. MA-OLSR benefits from using geo-information in both scenarios. However, the impact of using predicted information is highly depending on the channel conditions. The path planning approach requires access to recent mobility data of all nodes in network. Packet losses can lead to serious prediction errors and have a negative effect on the routing performance. The results of using the stigmergic prediction approach are shown in Fig. 8. In contrast to MA-OLSR, B.A.T.Mobile achieves a high robustness against packet losses. As each packet contains information about its whole travelled path, the routing process does not necessarily require data from all nodes of the network to make a decision. These properties demonstrate the protocol as well as the routing metric in general are highly benefitting from the interaction with the mobility control layer.

VI. Conclusion

In this paper, we presented an extension to the INET-MANET framework, which focuses on mobility-aware routing and uses a cross-layer approach to utilize mobility control information to enhance the routing process. The mobility behavior is implemented according to a well-known mobility meta-model that is able to provide a much more realistic representation of the mobility behavior of autonomous agents than generic algorithms. A new base module for mobility-aware protocols provides access to current and predicted mobility data as well as different routing metrics and information distribution methods. In a proof of concept evaluation, we demonstrated its potential to enhance the communication robustness even under challenging channel conditions. Overall, the consideration of mobility data for routing decision has shown to be a worthwile topic for further research and development. After the introduction of mobility-aware routing protocols, we want to extend the capabilities of the framework with communication-aware mobility algorithms. Furthermore, we want to add support for further vehicle classes. In this context the application and evaluation of the proposed routing protocols in the field of vehicular ad-hoc networks (VANETs) is another promising research topic.

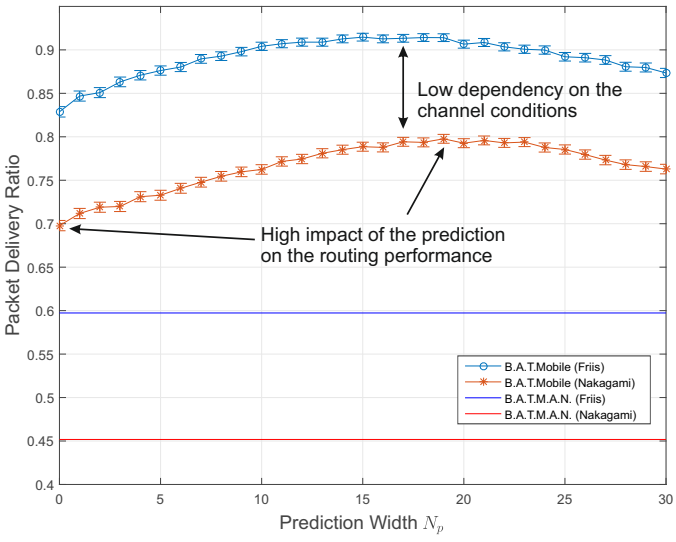


Fig. 8. Performance evaluation of B.A.T.M.A.N. using predictive stigmergy

Acknowledgment

Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project B4.

References

[1] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ser. Simutools '08. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 60:1–60:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1416222.1416290

[2] B. Sliwa, D. Behnke, C. Ide, and C. Wietfeld, "B.A.T.Mobile: Leveraging Mobility Control Knowledge for Efficient Routing in Mobile Robotic Networks," in *IEEE GLOBECOM 2016 Workshop on Wireless Networking, Control and Positioning of Unmanned Autonomous Vehicles (Wi-UAV)*, Washington D.C., USA, Dec 2016, accepted for presentation. [Online]. Available: http://arxiv.org/abs/1607.01223

[3] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," Internet Requests for Comments, RFC Editor, RFC 3626, October 2003, http://www.rfc-editor.org/rfc/rfc3626.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3626.txt

[4] D. Johnson, N. Ntlatlapa, and C. Aichele, "A simple pragmatic approach to mesh routing using batman," in *In 2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries, Pretoria, South Africa*, 2008.

[5] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, Feb 1999, pp. 90–100.

[6] M. Sbeiti, N. Goddemeier, D. Behnke, and C. Wietfeld, "PASER: Secure and Efficient Routing Approach for Airborne Mesh Networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 3, pp. 1950–1964, March 2016.

[7] M. Bahr, "Proposed Routing for IEEE 802.11s WLAN Mesh Networks," in *Proceedings of the 2Nd Annual International Workshop on Wireless Internet*, ser. WICON '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1234161.1234166

[8] Z. J. Haas, M. R. Pearlman, and P. Samar, "The Zone Routing Protocol (ZRP) for Ad Hoc Networks," Published Online, IETF MANET Working Group, INTERNET-DRAFT, Jul. 2002, expiration: January, 2003. [Online]. Available: http://www.ietf.org/proceedings/02nov/I-D/draft-ietf-manet-zone-zrp-04.txt

[9] B. Karp and H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '00. New York, NY, USA: ACM, 2000, pp. 243–254. [Online]. Available: http://doi.acm.org/10.1145/345910.345953

[10] R. Shirani, M. St-Hilaire, T. Kunz, Y. Zhou, J. Li, and L. Lamont, "Combined Reactive-Geographic routing for Unmanned Aeronautical Ad-hoc Networks," in *2012 8th International Wireless Communications and Mobile Computing Conference (IWCMC)*, Aug 2012, pp. 820–826.

[11] M. Mauve, J. Widmer, and H. Hartenstein, "A survey on position-based routing in mobile ad hoc networks," *IEEE Network*, vol. 15, no. 6, pp. 30–39, Nov 2001.

[12] C. W. Reynolds, "Steering behaviors for autonomous characters," *Game developers conference*, 1999. [Online]. Available: http://www.red3d.com/cwr/papers/1999/gdc99steer.html

[13] ——, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. ACM Press, 1987, pp. 25–34.

[14] J. Ruchti, R. Senkbeil, J. Carroll, J. Dickinson, J. Holt, and S. Biaz, "Unmanned Aerial System Collision Avoidance Using Artificial Potential Fields," *Journal of Aerospace Information Systems*, 2014.

[15] N. Goddemeier, S. Rohde, J. Pojda, and C. Wietfeld, "Evaluation of Potential Fields Mobility Strategies for Aerial Network Provisioning," in *Proc. of IEEE Globecom Workshop on Wireless Networking for Unmanned Autonomous Vehicles*. Houston, TX, USA: IEEE, Dec. 2011.

[16] D. Behnke, P.-B. Bök, and C. Wietfeld, "UAV-based Connectivity Maintenance for Borderline Detection," in *IEEE 77th Vehicular Technology Conference (VTC-Spring)*, Dresden, Germany, June 2013.

Implementation of the SWIM Mobility Model in OMNeT++

Asanga Udugama[1], Behruz Khalilov[1], Anas bin Muslim[1], Anna Foerster[1], Koojana Kuladinithi[2]

[1]Sustainable Communications Networks Group, University of Bremen, Germany
Email: {adu | bk | anas1 | afoerster}@comnets.uni-bremen.de
[2]Communications Networks Group, Hamburg University of Technology, Germany
Email: {koojana.kuladinithi}@tuhh.de

***Abstract*—The Internet of Things (IoT) is expected to grow into billions of devices in the near future. Evaluating mechanisms such as networking architectures for communications in the IoT require the use of simulators due to the scale of the size of networks. Mobility is one of the key aspects that need to be evaluated when considering the different scenarios of the IoT. Small Worlds in Motion (SWIM) is a mobility model that mathematically characterises the movement patterns of humans. The work presented in this paper details the development and verification of the SWIM mobility model in the OMNeT++ simulator.**

I. Introduction

The amount of devices in the Internet of Things (IoT) is expected to be over 200 billion by 2020 [1]. Peer-to-peer opportunistic networking is a model that is being considered for communications in the IoT [2]. When considering usage scenarios of the IoT based devices, many scenarios relate to human oriented activities (e.g., health monitoring). When the human element is involved, mobility of devices become an inherent characteristic of any scenario (e.g., a Smartphone user who has enabled health monitoring may travel to work daily).

When evaluating the performance of models for communications in the IoT, use of simulators such as OMNeT++ become a necessity due to the scale of the networks. The myriad of devices that form the IoT require appropriate models to correctly evaluate the performance. One such modeling area is mobility.

The OMNeT++ simulation environment provides implementations for a number of mobility models. Though these mobility models were developed to address different requirements of simulating mobile devices, [3] has shown through empirical evaluations that mobility patterns of humans are much more different than what these models model. Therefore, to cater to this disparity, the authors of [3] have proposed a simple self tuning mathematical model to model human behaviour with respect to mobility.

This model, named Small Worlds In Motion (SWIM) [3] uses two intuitions of human mobility, viz., people usually visits mostly locations close to their home location and if they visit a location far from home, it is due to its popularity. The authors of [3] have characterised these intuitions in a simple equation that considers the distance to a location and the popularity of that location. The work presented in this paper provides a description and the verification of the implementation of the SWIM mobility model in OMNeT++.

The rest of this paper is ordered in the following manner. The next section (Section II) provides a brief introduction to the classifications of the mobility models already supported in OMNeT++ and the implementation architecture of mobility in OMNeT++. Section III provides a description of the SWIM mobility model highlighting the most relevant parts required to implement the model in OMNeT++. Section IV details the implementation of the SWIM model in OMNeT++. Section V is a validation of the performance of the model in terms of the selections made of destinations. Section VI is a concluding summary.

II. Mobility Models and Mobility Implementation Architecture in OMNeT++

Network simulators employ two methodologies to simulate node mobility, viz., synthetic mobility models and traces [4]. Synthetic models are mathematical models that provide a list of coordinates for a node to determine its movement pattern. Traces are coordinate data collected from actual movement patterns of nodes that could be fed into a simulator to simulate mobility.

OMNeT++ provides a number of implementations for synthetic models and trace based mobility. All these implementations extend the functionality of a set of base classes related to handling mobility in OMNeT++. The synthetic mobility models in OMNeT++ are further classified into models with deterministic movements and models with random movements as described below.

A. Synthetic Mobility Models

Deterministic Movement Models - Models based on deterministic movements have the characteristic of the same movement pattern without any stochasticity. An example is the *Linear Mobility Model* where a node is made to move at a constant speed within a pre-determined mobility area. When the node reaches a border of the mobility area, it deflects at a pre-determined angle to continue with it's mobility.

Random Movement Models - Models with random movements employ stochastic movement patterns to move a node within a given area. An example is the *Random Way-point Mobility Model* where a node moves in segments within the mobility area. The speed and the direction for each segment are uniformly distributed random numbers.

The SWIM model implemented in this work is a synthetic model with a random movement direction determined by the SWIM equation (described in Section III).

B. Mobility Implementation Architecture in OMNeT++

OMNeT++ provides an interface (i.e., Abstract Class) that any mobility mechanism (model or trace) must implement to enable mobility in a node. This interface, called *IMobility* provides a set of methods that is invoked from other models that require mobility related information. An example is the wireless propagation models which require to know current coordinates, speed of mobility, etc. to determine the connectivity to other nodes.

To simplify the implementation of mobility, a set of base implementations are provided in OMNeT++ that implements some of the abstract methods of *IMobility*. These base implementations focus on implementing the basic functionality required in mobility patterns that could be generalised into categories. An example is the *LineSegmentsMobilityBase* class which is a general class to be extended when the pattern of mobility is linear and in segments within the mobility area. Similarly, the *LinearRotatingMobilityBase* provides the basic functionality for a node that is required to move in a circular manner within the mobility area.

The SWIM implementation presented in this work extends the *LineSegmentsMobilityBase* class.

III. Small Worlds in Motion (SWIM)

Small Worlds In Motion (SWIM) [3] is a simple mobility model meant for efficient simulation of human movements. The model is based on the following two intuitions of human movements.

- A visited location is either near to a person's home location;
- or, if the visited location is far from the home location, it is visited due to the popularity of that location.

The basis for characterising human mobility as itemised above is due to the nature of human mobility as explained below.

In daily life, people travel more often to places that are near to their homes or to places where they can meet other people. People rarely travel to places which are far from their home locations and if they decide to travel, the reason is due to the popularity of that location. When making travel decisions, it is usually a trade-off; Where to go? Either to go to the best supermarket or the most popular restaurant that is also not far from home location. Very rarely do people travel to a place that is far from home, or that is not popular.

These simple behavioural observations of human mobility provide the basis for SWIM which the authors have validated using real traces and the distributions of key parameters (viz., Inter-contact Time, Contact Duration and the number of contact distributions between nodes).

In the SWIM mobility model, each person (i.e., node) is assigned to a specific home location that is a randomly and uniformly chosen over the network area. Additionally, each node also assigns a weight to each of the possible destinations (i.e., locations). The weight is a function of the popularity of the location and the distance from home location; higher popularities influencing in a positive manner while higher distances influencing in a negative manner. In the SWIM model a network is divided into many small cells that represent possible locations. The size of the cells depend on transmitting range of mobile nodes in that cell. Each node builds a map based on this knowledge. A mobile node calculates the weight for each cell in the map. Once the weight for each cell is known, a node uses this information to choose the next destination to travel to. According to [3] the node chooses its destination cell randomly and proportionally considering the α and each node's weight, whereas the exact destination point within the chosen cell is taken uniformly at random over the cell's area. During the movement of a mobile node two characteristics will be followed. Firstly, the speed that a node travels from its current location to destination remains same. Secondly, the movement pattern is a straight line. When a node reaches a destination, it pauses for a period of pause time, chooses the next destination and continues the process. The process of selecting a destination is a two-step process.

1) Determine the next destination location type (i.e., a neighbouring location or a remote location). A constant called α is used to determine the destination type (i.e., an α probability of selecting a neighbouring location and a $1-\alpha$ probability of selecting a remote location);
2) Once the destination type is selected, the weight is used to determine the exact destination location from the type of destination. The equation for computing the weight is given below.

$$w(C) = \alpha \cdot distance(h_A, C) + (1 - \alpha) \cdot seen(C)$$

This equation represents the weight that node A assigns to cell C.

- $distance(h_A, C)$ is the distance from node A to cell C, and it decays based on power-law;
- α is a constant in the range of $[0; 1]$;
- $seen(C)$ is the number of nodes that node A encountered at cell C, and this value updates each time node A visits cell C.

The value of α influences the next destination chosen. If the value of α is large, then a mobile node is more likely to choose a destination near to it's home location while a small α results in the node selecting popular locations away from the home location.

The waiting time [3] in the SWIM model follows an upper bounded power law distribution (i.e., a truncated power law). This characterisation follows how people usually spend their time; most time at a few locations (e.g., workplace) and short times at many of the other locations (Pareto Principle).

The distribution of the inter-contact times show power law characteristics until a certain point of time after which it demonstrates an exponential cut-off. Experiments have proven [3] that the use of an appropriate α value results in the distribution of inter-contact times as in real life.

IV. SWIM Implementation in OMNeT++

The SWIM mobility model implemented for OMNeT++ extends the functionality in the INET framework. The INET framework provides the basic functionality required by SWIM in the following areas.

- Constant speed of mobility
- Linear movements

The *LineSegmentsMobilityBase* of OMNeT++ implements the functionality of linear movements at constant speeds to a given destination. The SWIM extensions implement the decisions on next destination selections. Figure 1 shows the procedure of the operation of the SWIM mobility model.

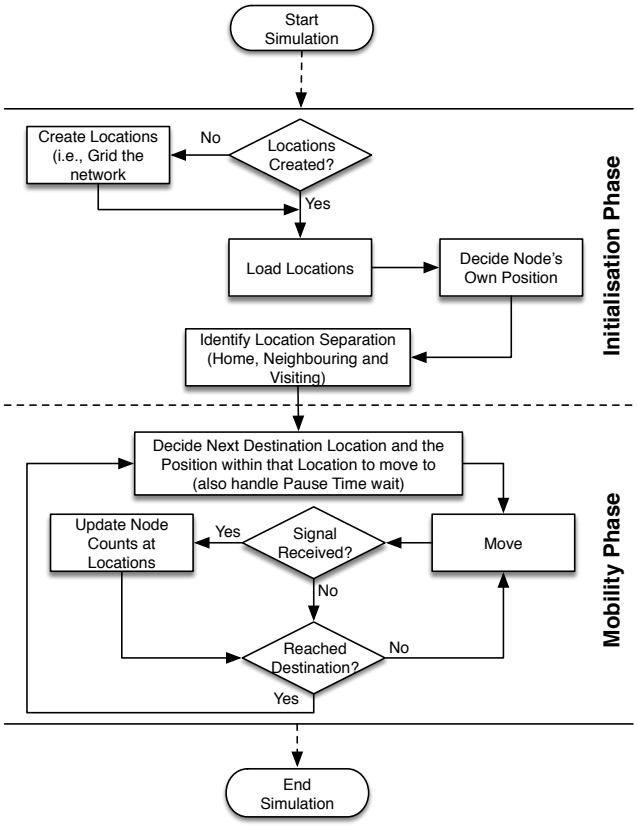


Fig. 1. Operation of the SWIM Mobility Model

The implementation has two parts to its operation, viz., *Initialisation Phase* and *Mobility Phase*. In the implementation, reference is made to two key terms; *Positions* and *Locations*. A position is the exact coordinates of a node at any given time. A location is an area where multiple nodes may be positioned.

A. Initialisation Phase

The *Initialisation Phase* is implemented in the *initialize()* method of the model implementation architecture of OMNeT++. The following tasks are performed in the initialisation.

Create and Load Locations - The network is identified by a cubic area within which all nodes will move (identified by the coordinates *constraintAreaX*, *constraintAreaY*, *constraintAreaZ*, *constraintAreaWidth*, *constraintAreaHeight* and *constraintAreaDepth*. This area is split into a grid of sub-areas which are identified as *Locations*. The size of each of these locations are equal and they correspond to the wireless range of the wireless technology used to communicate between the nodes. It is assumed, that all nodes in a single location is able to communicate with each other directly. These locations are common to all nodes in the network and therefore, the node that performs the *Initialisation Phase* first, creates a text file of these locations with their coordinates which are then loaded by all the other nodes in their corresponding *Initialisation Phases*.

Decide Node's Own Position - Every node has a starting position identified by the coordinates that lie within the mobility area (identified by *initialX*, *initialY* and *initialZ*). These initial position coordinates are uniformly distributed random numbers (i.e., coordinates) over the mobility area.

Identify Location Separation - Once a node's position is determined and all the locations are identified, the relevance of each location with respect to the node must be determined. The relevance falls into three categories.

- *Home* Location - Home location is the area of the grid where the node is located at the beginning of the simulation.
- *Neighbouring* Location - A neighbouring location is a location that is in the vicinity of a home location. Each node will have many neighbouring locations. The determination of neighbouring locations is done by using the *neighbourLocationLimit* model parameter. Any location that lies within the radius (distance given by *neighbourLocationLimit*) of a node's home location is considered as a neighbouring location.
- *Visiting* Location - If a location, does not fall into any of the above two categories, it is considered as a visiting location.

Parameter	Purpose
neighbourLocationLimit	Distance in meters used to determine the location separation (i.e., *Neighbouring* or *Visiting*)
speed	Speed of movements in meters per second
initialX, initialY, initialZ	Initial coordinates of the node
maxAreaX, maxAreaY, maxAreaZ	The area that the node may be allowed to move in, if such a restriction is required to be made
waitTime	The pause time of the node once it reaches a destination (before it starts moving to the next destination)
alpha	The α value used in the SWIM equation (see Section III)
noOfLocations	The number of locations to create from the movement area

TABLE I. Model Parameters

B. Mobility Phase

The actions performed in the *Mobility Phase* is implemented in the *setTargetPosition()* and the *move()* methods of the mobility related INET classes, and the *receiveSignal* method of the signal mechanism of OMNeT++. The SWIM implementation uses the signal mechanism of OMNeT++ to notify nodes about the arrival of a node at a certain location. The following tasks are performed in the *Mobility Phase*.

Decide Next Destination and Pause Time - When a node reaches a given destination, it has to pause for the specified time which is given with the *waitTime* model parameter. Before pausing, the node sends a signal to other nodes to indicate its arrival at a new location. Once the pause time is completed, the next destination to move to, is identified. The next destination is determined randomly but influenced by α value and the weights assigned to each of the destinations by a node. The equation in Section III computes the weight assigned to each location.

Move - The linear and constant speed movement of the node is handled by the *move()* method of the *LineSegmentsMobilityBase* class which is extended by the implementation.

Update Node Counts when Signal Received - When other nodes send signals indicating their arrival at a certain location, all nodes that are currently in that location must update their node counts to build their location popularity information. Nodes that are not at the location of the originator of the signal will ignore the signal as they have not "*seen*" (i.e., second part of the SWIM equation in Section III) the node.

Table I shows the configurable parameters of the model which could have static values or based on distributions.

V. Model Verification

The performance verification of the movement patterns of the developed SWIM model is done using the example mobility setup in INET. Figure 2 shows the movement pattern of a single mobile node and with an α set to 0.3. The model has identified 21 locations to move between and one of them is the home location. Based on the distance (determined by the *neighbourLocationLimit* model parameter), 16 locations are considered as neighbouring locations while the rest of the locations are considered as visiting locations. As explained before (in Section III), the α value influences the likeliness of the next destination selection. Since the α is a lower value (i.e., 0.3), selection of destinations is skewed towards selecting more remote locations (i.e., visiting locations) than neighbouring locations.

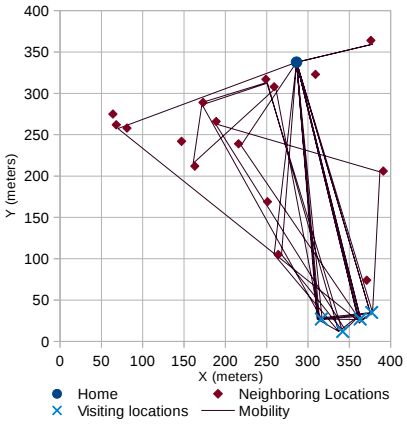


Fig. 2. Movement Pattern of a Node with an α of 0.3

Figure 3 shows the movement pattern of the same scenario (as in Figure 2) but with an α of 0.8. Since the α is higher, the location selection is skewed towards selecting more neighbouring locations than remote locations.

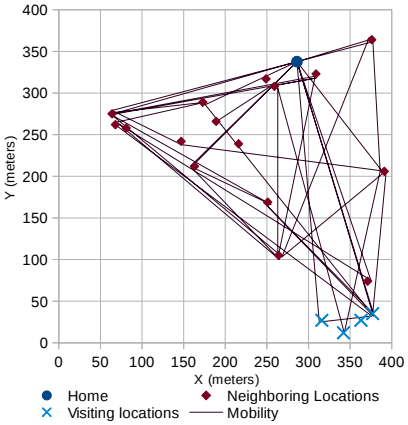


Fig. 3. Movement Pattern of a Node with an α of 0.8

Table II lists the parameters used in obtaining the above shown results. The mobility area was specified using 2 dimensions (i.e., *initialZ* and *maxAreaZ* were set to 0).

Parameter	Value
neighbourLocationLimit	300 meters
speed	1.4 meters per second
initialX, initialY	Randomly selected values
maxAreaX, maxAreaY	400 meters x 400 meters
waitTime	Randomly selected value between 2 and 5 seconds
alpha	0.3 and 0.8
noOfLocations	21 locations

TABLE II. Used Parameter Values

VI. Conclusion

The work presented in this paper relates to the development of the SWIM [3] mobility model in OMNeT++ to simulate human behaviour based movements. The work involved in understanding the mathematical model behind the operation of SWIM, understanding the mobility implementation architecture of OMNeT++, implementing the SWIM model in OMNeT++ and finally, verifying the performance.

The verification shows the influence of the differing α values on the location selection process. A larger α value results in a simulation selecting more of the neighbouring locations and less of the remote (i.e., visiting) locations. On the other hand, a smaller α value results in the vise-versa of location selections.

The SWIM mobility model developed in this work is released at Github (*https://github.com/ComNets-Bremen/SWIMMobility*) under a GPL License.

References

[1] International Data Corporation (IDC) IVIEW, *Digital Universe in 2020*, December 2012.

[2] A. Foerster et al, A Novel Data Dissemination Model for Organic Data Flows, MONAMI 2015, September 1618, 2015, Santander, Spain

[3] A. Mei and J. Stefa, *SWIM: A Simple Model to Generate Small Mobile Worlds*, IEEE INFOCOM, 2009.

[4] T. Camp and J. Boleng and V. Davies, *A Survey of Mobility Models for Ad Hoc Network Research*, Wireless Communications and Mobile Computing, August 2002

Babel Routing Protocol for OMNeT++

More than just a new simulation module for INET framework

Vladimír Veselý, Vít Rek, Ondřej Ryšavý
Department of Information Systems, Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
{ivesely, rysavy}@fit.vutbr.cz; rek@kn.vutbr.cz

***Abstract*—Routing and switching capabilities of computer networks seem as the closed environment containing a limited set of deployed protocols, which nobody dares to change. The majority of wired network designs are stuck with OSPF (guaranteeing dynamic routing exchange on network layer) and RSTP (securing loop-free data-link layer topology). Recently, more use-case specific routing protocols, such as Babel, have appeared. These technologies claim to have better characteristic than current industry standards. Babel is a fresh contribution to the family of distance-vector routing protocols, which is gaining its momentum for small double-stack (IPv6 and IPv4) networks. This paper briefly describes Babel behavior and provides details on its implementation in OMNeT++ discrete event simulator.**

Keywords—Babel, OMNeT++, INET, Routing, Protocols, IPv6, IPv4, dual-stack;

I. Introduction

Currently deployed routing protocols, e.g., OSPF, RIP, EIGRP, have well-known characteristics. New proposals that address shortcomings of standard routing protocols need to be evaluated to reach the required level of maturity to be widely adopted by the industry. Simulation approach can provide an initial evaluation of a new routing protocol and its comparison to existing protocols. Presented paper provides notes on development and evaluation of a simulation model for Babel routing protocol.

The newly developed Babel simulation model is a part of ANSAINET framework[1] which aims at providing a variety of simulation models compatible with RFC specifications and reference implementations. These tools should allow for analysis of a near real network behavior. Results are freely available and can be employed for further research initiatives, such as simulation approach to proving (or disproving) certain aspects of technologies and related protocols. The ANSAINET now supports:

- Babel dynamic unicast routing protocol;
- Proprietary first-hop redundancy protocols (FHRP) such as Hot Standby Router Protocol (HSRP) and Gateway Load Balancing Protocol (GLBP), which guarantee high-availability of default gateway;
- Device discovery protocols such as Cisco Discovery Protocol (CDP) and Link Layer Discovery Protocol (LLDP), which verify data-link layer operation.

In this paper, we only focus on a Babel simulation model. Babel is increasingly more popular seen as the open-source alternative to Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP). Babel is also considered a better routing protocol for mobile networks comparing to Destination-Sequenced Distance-Vector (DSDV) or Ad hoc On-Demand Distance-Vector (AODV) routing protocols. Babel is a *hybrid distance vector* routing protocol. Although it stems from a classical distributed Bellman-Ford algorithm, it also adopts certain features from link-state protocols, such as proactive neighbor discovery. It offers a great modularity of metric calculation (currently four distinct techniques are specified) and pluggable best route selection policy. Babel employs a *feasibility condition test* to prevent route loops during convergence phase. Babel protocol syntax employs type-length-value (TLV) encoding, which for instance allow incorporating different address-family (currently both IPv4 and IPv6) for routing information. Babel has a built-in mechanism for duplicity prevention together with a data compression that reduces protocol overhead.

This paper has the following structure. Section II covers a quick overview of existing implementations. Section III describes the operational theory and design supplemented with implementation notes. Section IV contains validation and testing scenarios. The paper is summarized in Section V together with outlines of our plans.

II. Current Babel Implementations

This section brings brief information about the availability of Babel for real network deployment. According to the main Babel project web page [3], there are three active (and one deprecated) implementations available:

- *babeld* – reference implementation maintained by the author of Babel specification [4];
- *Pybabel* – implementation in Python limited only to IPv6 and no real cost recalculation [5];
- *Sbabeld* – a limited implementation intended only for IPv6 stub-routers [6];

[1] ANSA is a long-term project carried by researchers and students at the Brno University of Technology aiming at extending IP network simulation framework with new simulation models. The framework is built on the original INET framework [1] shipped with OMNeT++ simulator.

We have unsuccessfully searched for any Babel simulation model. In particular, we are not aware of any Babel implementation in the most used discrete event simulators, such as NS-2/3, OPNET, or OMNET++.

III. Contribution

The aim of this section is to overview the basic elements of Babel: 1) the best route selection, 2) message exchange mechanisms, and 3) state maintenance structures. Also, we provide an explanation of how these elements are implemented in Babel's simulation model of ANSAINET.

A. Theory of Operation

Babel is codified withing IETF as experimental RFC 6126 [7]. It leverages both unicast communication and also multicast address 224.0.0.111 for IPv4 and ff02::1:6 for IPv6 group communication. Babel operates over UDP on (both source and destination) port 6696.

Babel is using **feasibility condition (FC)** when verifying incoming routing records. In particular, Babel employs FC variant called *Source Node Condition* [8] just as EIGRP: The best known metric m_A together with a sequence number s_A (number reflecting age of metric, higher means younger and more current) to a destination network $\boldsymbol{N}$ from a router $\boldsymbol{A}$ denotes its feasible distance, $FD_A(\boldsymbol{N}) = (s_A, m_A)$. Routing information received by router $\boldsymbol{A}$ from router $\boldsymbol{B}$ satisfies FC if and only if the metric $D_B(\boldsymbol{N})$ to the destination network $\boldsymbol{N}$ advertised by router $\boldsymbol{B}$ is strictly lower than $FD_A(\boldsymbol{N})$:

$$D_B(\boldsymbol{N}) = (s_B, m_B), FD_A(\boldsymbol{N}) = (s_A, m_A):$$
$$D_B(\boldsymbol{N}) < FD_A(\boldsymbol{N}) \leftrightarrow (s_B = s_A \wedge m_B < m_A) \vee s_B > s_A$$

Applying FC, we avoid counting-to-infinity problem known from original RIP implementation. However, the FC might cause *starvation*, when the only one route exists, and it cannot be used because it does not satisfy FC. Therefore, Babel is checking sequence numbers (just as DSDV) to recognize outdated FD. Moreover, Babel equips routing updates also with advertising router identification to distinct between different routes to the same network prefix.

Babel employs the sum of links costs from the router to a given destination network as the metric. Babel evaluates metric on router $\boldsymbol{A}$ as the route's metric m_B announced by a neighbor $\boldsymbol{B}$ plus link's cost c between $\boldsymbol{A}$ and $\boldsymbol{B}$: $m_A = m_B + c$ Babel currently suggests two methods how link's cost may be calculated:

- *k-out-of-j* – This method is intended for wired networks using two parameters j and k, where $0 < k \leq j$. A router remembers window of size j containing last k *Hello* messages. If less than k *Hello* messages have been delivered, then cost is set to 0xFFFF, which means infinity (unreachable network), otherwise (k and more *Hellos* have been successfully delivered) cost is set to a fixed value reflecting the link's speed (by default 96 on wired and 256 on wireless interfaces);
- *ETX* (based on [9]) – This method targets specifics of wireless networks. Link cost varies in time, and it is determined based on two parameters – successful *Hello* reception (β) and successful *Hello* transmission (α). Aggregated link cost is computed as $256/(\alpha \cdot \beta)$.

Babel exchanges data as TLV records. Babel message consists of several TLVs records, which is controlled by a buffering policy. Babel records are:

- *AckReq* and *Ack* – *AckReq* is the ack request, and *Ack* is the solicited acknowledgment response within specified interval using the same nonce;
- *Hello* – Neighbor and link's reception cost are discovered using *Hello*;
- *IHU – I Hear You* is confirmation of mutual reachability of neighbors. Moreover, these TLVs carry link's transmission cost;
- *Router-Id* – TLV contains unique (recommended is to use EUI-64) eight bytes long router identification within a given Babel routing domain;
- *NextHop* – TLV carries next-hop IP address for subsequent *Updates* TLV;
- *Update* – It advertises or withdraws routes including their prefix, two bytes long sequence number and metric;
- *RouteReq* – TLV prompts receiver to send *Update* regarding specified prefix;
- *SeqNoReq* – It prompts receiver to send (or to delegate further) *Update* regarding specified prefix with a given sequence number.
- *Pad1* and *PadN* – These two are padding TLVs without any meaningful content;

Babel standard specifies multiple abstract data structures and placeholders for information important for Babel functionality:

- *Interface Table* contains the list of interfaces through which Babel routing information are being sent and accepted;
- *Neighbor Table* (NT) contains the list of all known neighbors including mutual interface (through which neighbor is reachable), IP address, history of *Hellos*, transmission cost, sequence number (used by neighbor);
- *Source Table* (ST) is a placeholder for *FD*s of different network prefixes. Each record is specific for a given advertising router and prefix (including prefix length) and contains (s, m) tuple;
- *Route Table* (RT) contains Babel routes (tuples of prefix, prefix length, next-hop, neighbor's router-id and address, metric, sequence number) known to this router;
- *Pending Requests Table* (PRT) maintains the list of sent but not yet answered requests during network topology convergence time;

B. Simulation Model Implementation

In OMNet++, Babel is implemented as the compound module `BabelRouting` interconnected via `UDP` sockets with IPv4 and IPv6 network layers. It consists of three submodules

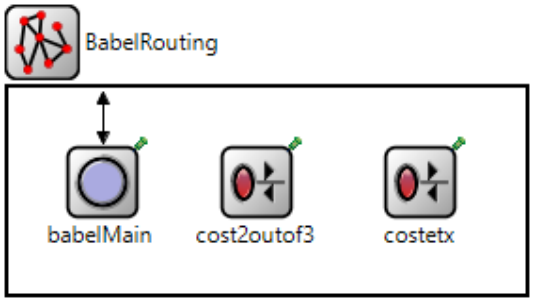


Fig. 1: BabelRouting module structure

that are depicted in Fig. 1 and briefly described in Table I. Our implementation is in full compliance with standard RFC 6126.

TABLE I. DESCRIPTION OF BABEL ROUTING SUBMODULES

Name	Description
`babel Main`	Implements Babel routing protocol behavior. Module contains auxiliary classes such as: `BabelInterfaceTable` (contains the list of Babel enabled interfaces including relevant settings); `BabelNeighborTable` (maintains the state of neighborship with adjacent routers); `BabelTopologyTable` (all routes towards known destinations); `BabelSourceTable` (table containing *FD*s, network prefixes and identifiers of routers propagating routing information); `BabelPenSRTable` (the list of pending requests); `BabelFtlv` (internal representation of uncompressed TLV records); and `BabelBuffer` (sending buffer for combining multiple TLVs into one outgoing message).
`cost 2outof3`	Implements link cost calculation employing method *k-out-of-j* with $k = 2$ and $j = 3$. This algorithm is usually used in wired networks.
`cost etx`	Implements link cost calculation using method ETX, which is suitable for wireless networks.

The user may easily repeat the design pattern of `cost*` modules to create a new policy of link cost calculation.

Babel behavior relies on timers. Following eleven timers are fundamental and corresponds to various conditions:

- *Hello timer* specifies time between two consecutive *Hello* messages (by default 20 seconds on wired and 4 seconds on wireless links);
- After *Update timer* expiration, periodic *Updates* are sent (by default 4× *Hello timer*);
- *Buffer* and *BufferGC timers* are employed to remove (un)used data from buffer;
- *ToAckResend timer* governs retransmission of *Ack*;
- *NeighHello* and *NeighIHY* timers hold neighbor's *Hello* and *IHY* intervals;
- *RouteExpiry* and *RouteBefExpiry* affect each RT's records validity period;
- Each record in ST has its *SourceGC* timer; Babel removes the record from ST after the timer's expiration;
- *SRResend* timer triggers PRT's request resending.

IV. TESTING

In this section, we provide information regarding validation of our simulation model against existing implementations. The goal is to demonstrate that this module is accurate enough for simulation of real network scenarios.

For test cases, we have built the same *real* network topology as for the simulation. We captured and analyzed (using Wireshark 1.12.7 packet sniffer) relevant Babel messages exchanged between routers. The routers run *babeld* v1.5.1 on operating system Debian 7.7 x64 (kernel 3.2.65-1).

Topology, which is depicted in Fig. 2, contains four routers (`R1`, `R2`, `R3`, and `R4`), four Local Area Network (LAN) segments (either simulated using dual-stack host or directly connected interfaces) and interconnections between routers. Topology operates both IPv4 and IPv6 address families. Babel *2-out-of-3* strategy is employed for link cost calculation. *Hello interval* is set to 4 seconds, and split-horizon is activated on all interfaces.

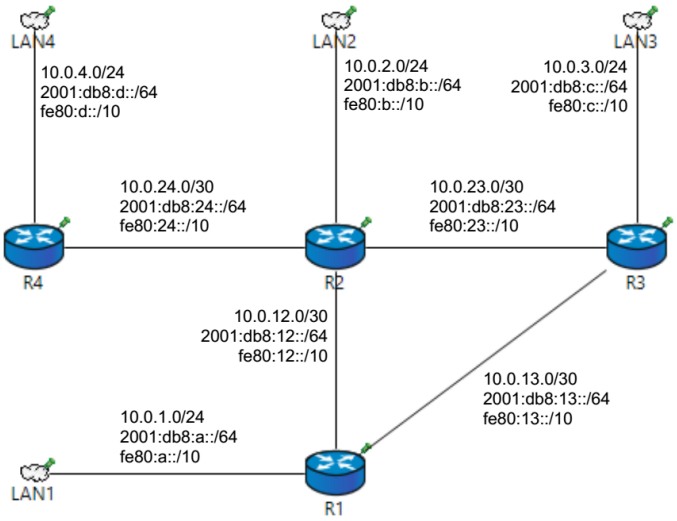


Fig. 2: Babel testing topology

We have conducted three tests to compare the behavior of implementation in common scenarios for Babel protocol: (A) establishing neighborship, (B) routing table convergence, and (C) link failure and subsequent routing information propagation. The details and results are described in the rest of this section.

A. Establishing Neighborship

This test aims at observing initial Babel message exchange between two adjacent routers when they discover each other on the link. We have decided to describe in more detail situation between routers `R1` and `R2` because establishing neighborship among any other two adjacent routers is analogous. We align events between simulation and real network by the time t_0 when router `R1` initiates Babel process while `R2` is already operational. We achieve this alignment: a) by executing *babeld* process in the real network; b) by connecting link between routers in the simulation. Table II summarizes exchanged TLVs. Column marked with "Ord." shows the order of intercepted messages. Column marked with header "S → R" contains sender

and receiver of a given message. Timestamps in "Simul." and "Real" columns are relative to the t_0 event.

TABLE II. ESTABLISHMENT NEIGHBORSHIP TIMESTAMP COMPARISON

Ord.	TLVs	S → R	Simul.[s]	Real [s]
#1	*Hello, RouteReq*	R1→R2	0.092	0.006
#2	*Hello, IHU, Update*	R2→R1	0.292	0.007
#3	*Hello, IHU*	R1→R2	0.492	0.040
#4	*Hello, IHU*	R2→R1	0.692	0.134
#5	*RouteReq*	R2→R1	0.692	0.903
#6	*Hello, IHU, Update*	R1→R2	0.892	1.084
#7	*RouteReq*	R1→R2	0.892	1.085
#8	*Update, IHU*	R2→R1	1.902	1.744
#9	*Hello, IHU*	R2→R1	5.632	5.111

Babel message #1 (containing *Hello* and *RouteReq* TLVs) announces `R1` to be present on the link. `R2` responds with #2 appending known routes in *Update* TLV. It takes #3 and #4 messages to pronounce neighbors as operational because of *2-out-of-3* strategy. `R2` sends #5 directly to `R1` via unicast asking `R1` for known routes. `R1` appends *Update* as the reaction to the previous message. `R1` repeats the similar process of route discovery via *RouteReq* in #7, where `R2` responds with #8. Expired *Hello timer* causes `R2` to send message #9.

Note that `babeld` can cope with lossy links by sending several copies of the same TLV (marked red in Table II). This advanced behavior not described in RFC and thus not implemented by the current version of the Babel simulation model.

B. Routing Table Convergence

During this test, we analyze network convergence process and compare the content of RTs in the resulting stable state. We focus on the IPv4 routes content in RT. Fig. 3 shows its content in the simulation and Table III in the real network. We can find the slight difference (marked with red) between simulated and real network. Destination network 2001:db8:23::/64 is reachable from `R1` via two routes – the first goes through `R2` and the second through `R3`. They have the same metric, and both of them are equally reachable. As Babel does not support load-balancing only one route can be selected. The selection of the route is implementation dependent. It explains why the simulation experiment differs to the real network experiment in this partial result.

TABLE III. R1'S ROUTE TABLE CONTENT IN THE REAL NETWORK

Flag Prefix	Met	RD	Router-Id	Next-Hop
> 2001:db8:a::/64	0			
> 2001:db8:12::/64	0			
> 2001:db8:13::/64	0			
> 2001:db8:b::/64	96	0	2222:2222:2222:2222	fe80:12::2
> 2001:db8:c::/64	96	0	3333:3333:3333:3333	fe80:13::3
> 2001:db8:d::/64	192	96	4444:4444:4444:4444	fe80:12::2
2001:db8:12::/64	96	0	2222:2222:2222:2222	fe80:12::2
2001:db8:13::/64	192	96	3333:3333:3333:3333	fe80:12::2
2001:db8:12::/64	192	96	2222:2222:2222:2222	fe80:13::3
2001:db8:13::/64	96	0	3333:3333:3333:3333	fe80:13::3
> 2001:db8:23::/64	96	0	2222:2222:2222:2222	fe80:12::2
2001:db8:23::/64	96	0	3333:3333:3333:3333	fe80:13::3
> 2001:db8:24::/64	96	0	2222:2222:2222:2222	fe80:12::2

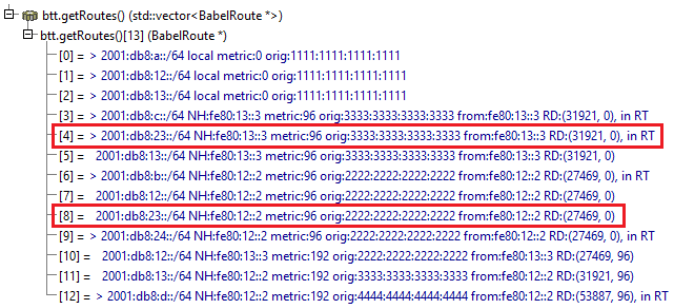


Fig. 3. R1's Route Table content in the simulation

C. Link-failure

We have scheduled link failure (physically disconnecting the link) between `R1` and `R2` at t_0. We analyze the impact on reachability of network 2001:db8:a::/64 from the perspective of router `R2`. Fig. 4 shows RT:

- before the failure (the next-hop is `R1` with address fe80:12::1 and metric 96);
- shortly after the failure (with poisoned metric 65535);
- after a while, when network correctly converges to `R3` as the new next-hop (with address fe80:23::3).

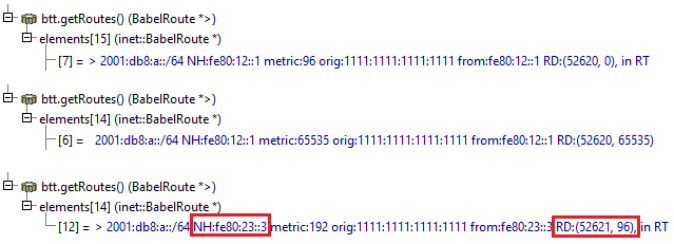


Fig. 4. R2's route state before/after the link failure and after convergence

This scenario demonstrates the usage of sequence numbers by Babel. If a backup route satisfying FC was available, then it would be immediately used. However, the poisoned route cannot be removed from the RT and updates from `R3` are ignored. Babel solves this starvation situation by incrementing route's sequence number. Related communication is outlined in Table IV, where timestamps are aligned with t_0 failure event.

TABLE IV. LINK FAILURE TIMESTAMP COMPARISON

Ord.	TLVs	S → R	Simul.[s]	Real [s]
#1	*SeqNoReq*	R2→R3	0.187	0.208
#2	*SeqNoReq*	R3→R1	0.347	1.079
#3	*Update*	R1→R3	0.595	1.152
#4	*Update*	R3→R2	0.673	1.275

Table IV reveals noticeable timestamp differences between simulated and real network environments in this test. Nevertheless, the order of messages is still preserved. Time variations are caused by three factors: a) *babeld* operation is influenced by operating system interrupts; b) built-in packet pacing avoiding potential race conditions; and c) inaccuracy in timing of t_0 event in the real network. Nevertheless, the routing tables in the simulated and real networks are equivalent.

V. Conclusion

In this paper, we presented a description of Babel dynamic routing protocol and its implementation as a simulation model for OMNeT++. We have designed and implemented a new simulation model mimicking the behavior of full-fledged Babel implementation for OMNeT++. Moreover, we have verified the accuracy of simulation results taking into account real wired network baselines.

Developed Babel simulation model is also a tool for the next steps of our future work. We plan to compare the properties of various routing protocols in specific scenarios (e.g., high availability data-centers). Another possibility would be to employ Babel in wireless scenarios but it is not our primary goal since ANSAINET is focused on traditional wired networks and their technologies. We hope that our contribution will be eventually included in the official INET release. Moreover, Babel module is also going to be employed in the frame of PRISTINE project [10] as a support of ongoing research.

All source codes could be downloaded from ANSAINET GitHub repository [11]. Real packet captures and simulation datasets for results reproduction could be obtained from Wiki pages of the above-mentioned repository. More information about ANSA project is available on its homepage [12].

Acknowledgment

This work was supported by the Brno University of Technology organization and by the research grants:

- FP7-PRISTINE supported by the European Union;
- FIT-S-14-2299 supported by Brno University of Technology;

References

[1] INET. (2016) INET Framework - INET Framework. [Online]. https://inet.omnetpp.org/

[2] OpenSim Ltd. (2015) OMNeT++ Discrete Event Simulator - Home. [Online]. https://omnetpp.org/

[3] Université de Paris-Diderot. (2016) Babel — a loop-avoiding distance-vector routing protocol. [Online]. http://www.pps.univ-paris-diderot.fr/~jch/software/babel/

[4] GitHub Babel. (2016) The Babel routing daemon. [Online]. https://github.com/jech/babeld

[5] M. Stenberg. (April, 2015) Babel implementation in Python3.4+. [Online]. https://github.com/fingon/pybabel/

[6] J. Chroboczek. (2015, November) Stub-only implementation of the Babel routing protocol. [Online]. https://github.com/jech/sbabeld/

[7] J. Chroboczek. (2011, April) RFC 6126: The Babel Routing Protocol. [Online]. https://tools.ietf.org/html/rfc6126

[8] J. J. Garcia-Lunes-Aceves, "Loop-Free Routing Using Diffusing Computations," IEEE/ACM Transactions on Networking, vol. 1, no. 1, pp. 130-141, February 1993.

[9] D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in MobiCom '03 Proceedings of the 9th annual international conference on Mobile computing and networking, New York, United States of America, 2003, pp. 134-146.

[10] PRISTINE consortium. (2016) PRISTINE | PRISTINE will take a major step forward in the integration of networking and distributed computing. [Online]. http://ict-pristine.eu/

[11] GitHub ANSA. (2016) ANSA extension above INET framework for OMNeT++. [Online]. https://github.com/kvetak/ANSA

[12] Brno University of Technology. (2016) ANSAWiki | Main / HomePage. [Online]. http://nes.fit.vutbr.cz/ansa/