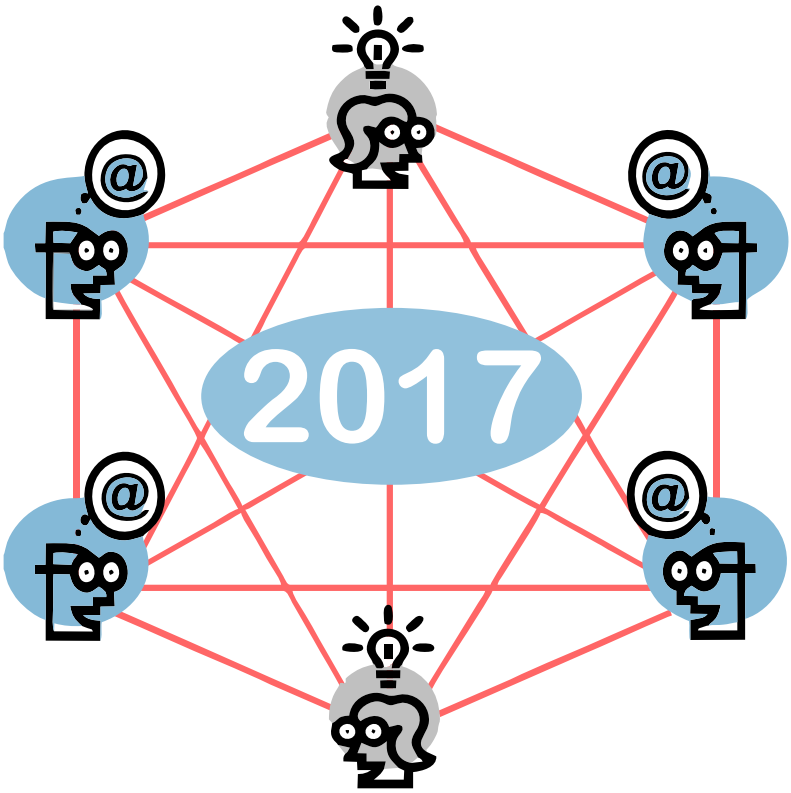
OMNeT++ Community Summit 2017

Successor of the International Workshop on OMNeT++

University of Bremen – Germany – September 07-08, 2017



Proceedings of the 4th OMNeT++ Community Summit

Community Summit Organizers

Founding Chair

Andras Varga (OpenSim Kft.)

Community Summit Organizers

Anna Förster (University of Bremen, Germany)

Asanga Udugama (University of Bremen, Germany)

Andreas Könsgen (University of Bremen, Germany)

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)

Vladimír Veselý (Brno University of Technology, Czech Republic)

Technical Program Committee

Claudia Campolo (University of Reggio Calabria, Italy)

Laura Marie Feeney (Uppsala University, Sweden)

Michael Frey (HU Berlin, Germany)

Florian Kauer (TU Hamburg-Harburg, Germany)

Marcel Marek (University of Oslo, Norway)

Yutaka Matsubara (Nagoya University, Japan)

Cyriel Minkenberg (Rockley Photonics Inc., USA)

Thi Mai Trang Nguyen (UUPMC - LIP6, France)

Christoph Sommer (University of Paderborn, Germany)

Mirko Stoffers (RWTH Aachen University, Germany)

Marco Tiloca (RISE SICS, Sweden)

Welcome Note from the OMNeT++ Community Summit Organizers

The fourth edition of the OMNeT++ Community Summit took place in September 2017 at the University of Bremen in Germany. The Free Hanseatic City of Bremen is the second biggest city in Northern Germany and a major economic and cultural hub with a long and rich history.

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 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 and the extension to a 2-day time frame allows us to better encompass more visionary ideas and foster interaction between the participants. The first three summits – 2014 in Hamburg Germany, 2015 in Zurich Switzerland, and 2016 in Brno Czech Republic – were huge successes with a vibrant program of keynotes, tutorials, discussion panels, demonstrations, and presentations about scientific research and industrial applications. In 2017, we continued to support the interactive aspects with tutorials and a discussion session with the OMNeT++ core developers. We also continued to peer-review all submissions to increase the scientific background and to give submitters feedback on their novel ideas and works in the context of modeling and simulating with OMNeT++.

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, wireless networks, overlay networks, and vehicular 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 community summit.

A glance at the technical program of the fourth Community Summit shows that a wide range of submissions with propositions of new simulation models and simulators as well as extensions of OMNeT++ and combinations of well-known extension frameworks were submitted. Submission topics range from aspects like mobility modeling to application fields like the integration of cellular communication in vehicular networks. We have gathered all individual contributions of the 2017 Summit in the form of proceedings to enable future reference and facilitate further discussion within the community.

We want to thank our local hosts – the Communication Networks Working Group at the University of Bremen – for providing us with everything necessary to realize an event like the Community Summit without the need to gather attendance fees. We thank our Publicity Organizers, Kyeong Soo (Joseph) Kim and Vladimír Veselý, 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 2017 for their work, attendance, and input. You are the quintessence for the success of the Community Summit.

Anna Förster (University of Bremen)
Asanga Udugama (University of Bremen, Germany)
Andreas Könsgen (University of Bremen, Germany)

Community Summit Organizers

Antonio Virdis (University of Pisa, Italy)
Michael Kirsche (BTU Cottbus-Senftenberg)

Technical Program Organizers

Contents

LIMoSim: A Lightweight and Integrated Approach for Simulating Vehicular Mobility with OMNeT++

Benjamin Sliwa, Johannes Pillmann, Fabian Eckermann and Christian Wietfeld
Communication Networks Institute
TU Dortmund
44227 Dortmund, Germany
e-mail: {Benjamin.Sliwa, Johannes.Pillmann, Fabian.Eckermann, Christian.Wietfeld}@tu-dortmund.de

***Abstract*—Reliable and efficient communication is one of the key requirements for the deployment of self-driving cars. Consequently, researchers and developers require efficient and precise tools for the parallel development of vehicular mobility and communication. Although current state-of-the-art approaches allow the coupled simulation of those two components, they are making use of multiple specialized simulators that are synchronized using interprocess communication, resulting in highly complex simulation setups. Furthermore, the compatibility of those simulators requires constant attention as they are developed independently. In this paper, we present a lightweight and integrated approach for simulating vehicular mobility directly in Objective Modular Network Testbed in C++ (OMNeT++) and INET without the need for external tools or Interprocess Communication (IPC). The proposed framework Lightweight ICT-centric Mobility Simulation (LIMoSim) is available as Open Source software and can easily be combined with other third-party extension frameworks for providing vehicular mobility based on well-known microscopical models. In contrast to existing approaches, the amount of necessary preprocessing steps for simulation setups is significantly reduced. The capabilities of LIMoSim are demonstrated by a proof of concept evaluation in combination with the Long Term Evolution (LTE) simulation framework SimuLTE.**

I. Introduction

OMNeT++ [1] is a well-established network simulation framework available as Open Source software for academic usage. Because of its modular approach, it has been extended by many third-party frameworks focusing on specialized communication technologies like LTE and IEEE 802.11p. As we have shown in previous work [2] the performance and robustness of communication systems can be highly improved by integrating knowledge of the users' mobility behavior into the routing decisions. With the deployment of autonomous cars in the near future, information about planned trajectories becomes available in all those vehicles and should be exploited for decision processes in the next generation of intelligent communication systems. With the increasing amount of interactions between mobility and communication, engineers and developers require efficient tools for simulating both aspects at once. Although current state-of-the-art frameworks already allow the coupling of specialized simulators for the two individual aspects, this approach has a number of system-immanent disadvantages (cf. Sec. II) because each of those simulators was developed for a very isolated field of application and not intended to be combined with others. In this paper, we present a lightweight framework for simulating microscopic vehicular traffic based on well-known models. In contrast to existing approaches, the proposed LIMoSim is intended to be used by communication simulators by design. It is seamlessly integrated into OMNeT++ (cf. Fig. 1) and requires no external tools or synchronization through IPC. Furthermore, it is completely compatible to third-party extension frameworks like SimuLTE [3] and INETMANET, which can integrate the novel mobility modules into their simulation scenarios in a transparent way using the widely-used INET framework. The rest of the paper is structured as follows. After discussing the related work, we present the basic architecture of our simulation model and provide a detailed description about the integration of LIMoSim into OMNeT++. In the next section, we describe the simulation setup for a proof of concept evaluation scenario in an LTE-context. Finally, detailed simulation results are presented and discussed.

II. Related Work

Vehicles in Network Simulation (Veins) [4] is a well-established simulation framework for simulating Vehicular Ad-hoc Networks (VANETs) in OMNeT++. It provides an implementation of IEEE 802.11p and acts as an interface to the microscopic traffic simulator Simulation of Urban Mobility (SUMO) [5]. Both simulators are synchronized through IPC using Transmission Control Protocol (TCP) and the dedicated Traffic Control Interface (TraCI) protocol of SUMO. While this approach ensures highly precise simulation results for both communication and vehicular mobility through usage of specialized simulators, it has a number of disadvantages:

- Since the different simulators are developed individually, their *compatibility* needs to be validated with every new

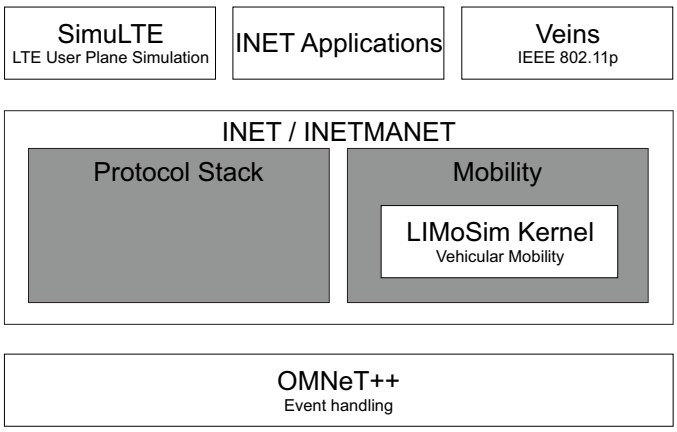


Fig. 1. Integration of the proposed framework into the family of OMNeT++ extensions. Since the the kernel of LIMoSim is embedded into the INET mobility module, the integration is transparent for the application-specific extensions frameworks like SimuLTE and can be used without requiring additional adjustments.

release. When we started to work on LIMoSim, Veins was not compatible to the newest SUMO version.
- For mobility-aware communication applications, TraCI is the bottleneck for the development process. The protocol needs to be extended for every new information type that should be shared between the simulators.
- Simulation setups have a high complexity because different tools have to be executed simultaneously. This aspect is even more dramatic for massive simulation scenarios that are executed on multiple servers in parallel. Additionally, even the generation of SUMO-only scenarios from OpenStreetMap (OSM)-data is quite complex, as several preprocessing steps are required.
- SUMO itself is rather designed for being used in a static way using precomputed data (for example routing paths). This does not match well with highly-dynamic vehicular applications, where the mobility behavior influences communication processes and vice versa.

Since the IPC-approach using SUMO is the current state-of-the-art way for simulating vehicular mobility in OMNeT++, those disadvantages are propagated to third-party extension frameworks like INETMANET and SimuLTE as well, if they want to make use of vehicular motion.

Consequently, we think the OMNeT++ community could benefit from a more diverse way for simulating vehicular mobility depending on the application scenario. While the current approach using Veins with SUMO is fitting for many IEEE 802.11p applications, there are scenarios where Veins could be coupled with our proposed framework to avoid the SUMO overhead and the need for IPC. Additionally, the complexity of many LTE and Mobile Ad-hoc Network (MANET) scenarios can be significantly reduced by using LIMoSim only.

III. Integration of LIMoSim into OMNeT++

LIMoSim focuses on highly dynamic traffic scenarios where all decision processes and routes are determined at runtime. Its main field of application is the simulation of medium-sized city scenarios, where intelligent vehicles interact with other traffic participants through means of communication. In this context, the simulation of vehicular mobility is considered as a service for the simulation of communication systems.

The simulator consists of two main components: the simulation kernel with the different elements of the microscopic mobility models and the User Interface (UI) part for standalone vehicle simulation and the road editor for easy generation of new mobility scenarios. It furthermore features live visualization of statistical data in dynamically updated plots and contains export functions for vector graphics. In this paper, we focus on the simulation kernel of LIMoSim, as it is the only component that is linked to OMNeT++ and provide descriptions of the hierarchical mobility model and the event handling mechanism in the following subchapters.

A. Simulation of Vehicular Mobility

The *LIMoSimCar.ned* module extends the *MovingMobilityBase.ned* module of the INET framework and acts as a logical structure for the different mobility-related submodules. Its hierarchical structure is illustrated in Fig. 2. A strategic model is used to determine the current destination point, which is the motivation for the vehicles movement. This node is

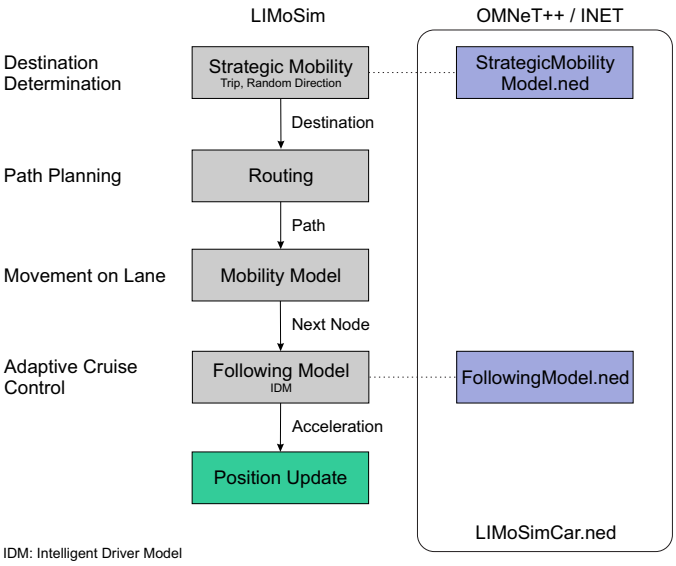


Fig. 2. Hierarchical model for microscopic traffic simulation encapsulated in the *LIMoSimCar.ned* module. OMNeT++-interface modules are available for the top and the bottom layer of the model.

then passed to a routing algorithm that computes the optimal path with respect to a defined criterion. All nodes of the path are handled sequentially by the general mobility model. Required adjustments to the mobility behavior because of encounters with other traffic participants are handled by the *Following Model*, which controls the distance between vehicles depending on their velocity by acceleration and deceleration. Finally, the position is updated with the calculated speed. In the current version of LIMoSim, we use the Intelligent Driver Model (IDM) [6] as a well-known following model for providing a realistic representation of vehicular mobility and combine it with the dedicated *lane-changing model* Minimizing Overall Braking Induced by Lane change (MOBIL) [7]. It should be noted that since the regular version of this model is not intersection-aware, the IDM implementation of LIMoSim treats traffic signals like static vehicles if the light is yellow or red. The integration of further models is planned for later releases. Since the individual mobility control modules

```
*.ue.mobilityType = "LIMoSimCar"
*.ue.mobility.map = "map.osm"
*.ue.mobility.strategicModel = "Trip"
*.ue.mobility.strategicModel.trip = "677230875,
275672221,3569208993,477807"
*.ue.mobility.way = "337055293"
*.ue.mobility.segment = 4
*.ue.mobility.lane = 0
*.ue.mobility.offset = 1m
```

Fig. 3. Example .ini configuration for using LIMoSim in an LTE-scenario. The trip model routes the car to a sequential list of a destinations. The actual node ids can be obtained from the UI part of LIMoSim or the raw OSM-files. The configuration can be either done manually by setting the parameters of the mobility modules or automatically using a generated XML file.

are interfaced through a *Simple Module*, the respective parameters can be set for the *.ned-file* without requiring external configuration files. An example configuration is shown in Fig. 3, where a User Equipment (UE) is assigned with the `LIMoSimCar` mobility type and positioned on a specified

way segment. Furthermore, the `Trip` strategic mobility model is configured to sequentially approach multiple destination points (cf. Fig. 5). The ids originate from the OSM data model and can be obtained using UI of LIMoSim.

B. Embedding LIMoSim-Events into OMNeT++

The Discrete Event Simulation (DES)-coupling mechanism is illustrated in Fig. 4. Since LIMoSim can also be used in a standalone mode, its objects are not aware of their OMNeT++-environment. The integration into the event handling mechanism is performed by a virtual event queue that does not take OMNeT++ events into consideration. In contrast to the IPC-based approach, there is no need for real DES-synchronization, as only a single queue is used. If an event e is scheduled, the *event mapping singleton* creates a new OMNeT++ *cMessage* m and stores a map entry for original event. m is then inserted into the OMNeT++ event queue. Once the *handleMessage()* method for m is called, e is retrieved from the map entry and handled by the respective object. With this approach, the actual event handling in OMNeT++ is transparent for all LIMoSim objects. Moreover, LIMoSim objects can use the event handling mechanism of OMNeT++ without even requiring an actual OMNeT++ module. As a consequence, objects that only influence the vehicular traffic but do not contain communication modules (like interference traffic or traffic signals) do not need to be modelled in OMNeT++.

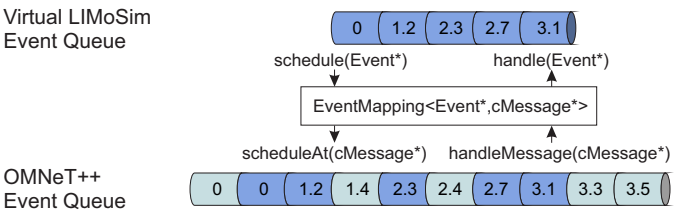


Fig. 4. Coupling of the event queues for mobility and communication simulation. LIMoSim objects are not aware of their execution environment and use a virtual event queue, which is mapped to the event queue of OMNeT++.

IV. Proof of Concept Evaluation

In this section, we present the setup and the results of the proof of concept evaluation. The default parameters for the scenarios are defined in Tab. I. Additional LTE-parameters are set according to the Handover (HO) example of SimuLTE.

TABLE I
Simulation Parameters of the Reference Scenario

Simulation parameter	Value
Strategic mobility model (UE)	Trip
Strategic mobility model (interference traffic)	Random Direction
Number of interference cars	100
Following model	IDM
Lane change model	MOBIL
Speed factor (driver behavior)	1 ± 0.2
Carrier frequency	1800 [MHz]
eNode B transmission power	46 [dBm]
eNode B antenna	omnidirectional

A. LTE-scenario with Real-world Map Data

As a realistic scenario, real-world map data from OSM is used. An LTE-enabled car is monitored while it is driving around the campus area of the TU Dortmund University using a *Trip* strategic mobility module. 100 other cars act as interference traffic with *Random Direction* mobility and influence the mobility behavior of the considered car. The area is covered by three different Evolved Node Bs (eNBs), which have been positioned according to network provider information. Fig. 5 provides an illustration of the described simulation setup. Due to the mobility of the car, multiple HOs are required at runtime. For the considered car, the current velocity, acceleration and the Received Signal Strength Indicator (RSSI) of the LTE link are measured over the trip duration and shown in Fig. 6.

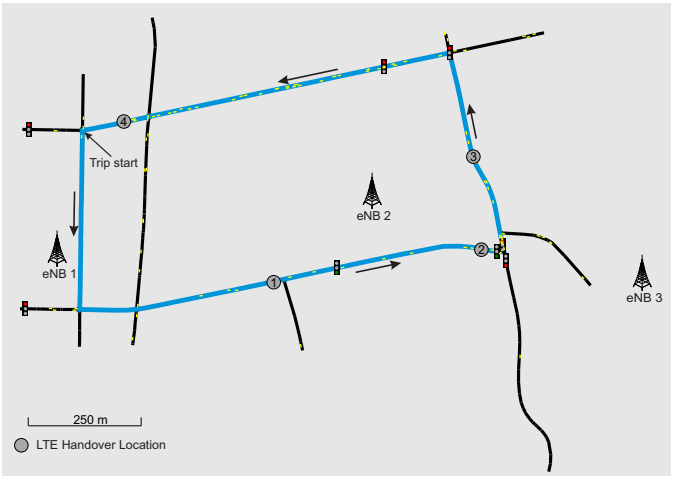


Fig. 5. Example scenario using OSM location data. 100 cars are simulated in the TU Dortmund University area. The map has been exported directly from the UI part of LIMoSim using its vector graphic export function.

The graphs for velocity and acceleration show typical inner-city characteristics and mirror the traffic dynamics of the scenario. Due to the map topology, the traffic signals and the other traffic participants, the current street condition is under constant change. Therefore, a lot of braking and accelerating is required, causing a highly dynamic velocity behavior. The RSSI of the LTE signal shows plausible characteristics depending on the distance to the current serving eNB. Four handovers occur during the considered time (for the actual handover locations cf. Fig. 5). With HO2, the device attaches from eNB2 to eNB3 and shortly afterward back to eNB2 with HO3. This *ping pong* behavior can be considered as a motivation for further research in the interdependency of communication and mobility, as the handovers could have probably been avoided using a mobility-aware decision approach. In earlier work [8], we have performed a similar case-study using a different toolchain with SUMO and an analytical LTE model, which achieved similar results for the handover behavior.

B. Example Behavior of the IDM model

In order to analyze the effects of the IDM model on the traffic dynamics, we consider a typical street scenario. Fig. 7 visualizes the acceleration behavior of multiple cars in a space-time diagram. The cars start in a jam situation, which is then

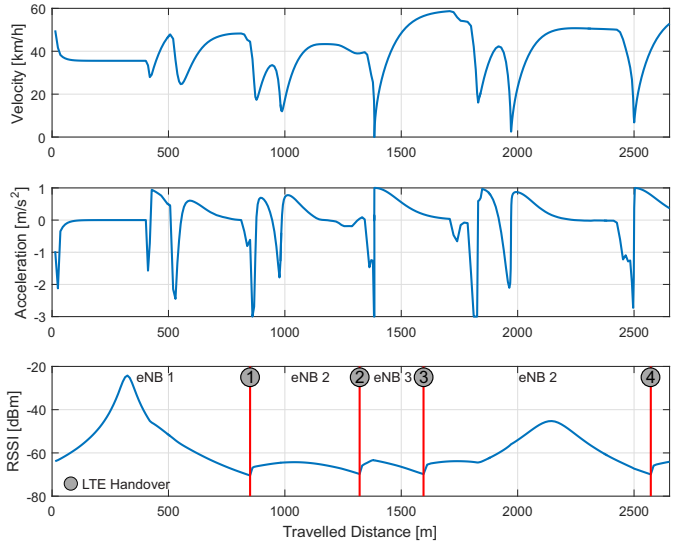


Fig. 6. Temporal behavior for velocity, acceleration and RSSI of the considered vehicle. The graphs show typical inner-city mobility characteristics and plausible LTE-behavior.

resolved into free traffic. After about 700m the cars encounter a car with engine failure that causes another traffic jam as only a single lane is used. Different driver behavior types can be identified depending on the slope of the curve. The results are confirmed by the analytical evaluation in [9].

V. Conclusion

In this paper, we presented the novel framework LIMoSim[1] for simulating microscopic vehicular mobility directly in OMNeT++. In contrast to existing approaches that treat vehicular mobility and communication separately and require IPC for the synchronization of different simulation tools, our proposal brings these aspects together in an integrated way. The actual mobility simulation relies on well-known models in order to guarantee the required accuracy. The easy integration is especially attractive for LTE and MANET simulations, which can make use of vehicular mobility without requiring a complex simulation setup. The capabilities of LIMoSim were demonstrated with a proof-of-concept evaluation in an LTE-context using real-world map data from OSM.

In future work, we want to couple LIMoSim with Veins for the simulation of IEEE 802.11p networks. LIMoSim could serve as an alternative to SUMO providing a lightweight solution for simulating vehicular motion without the IPC-overhead. Furthermore, we want to integrate strategic mobility models that are closer to human decision making like Small Worlds In Motion (SWIM) [10], which has already been applied to OMNeT++ in [11] and utilize the framework for the simulation of indoor robotic networks in a logistical context. Moreover, we want to exploit the visualization capabilities of the UI-part of LIMoSim to enable live visualization of OMNeT++ key performance indicators.

[1] Available at https://github.com/BenSliwa/LIMoSim

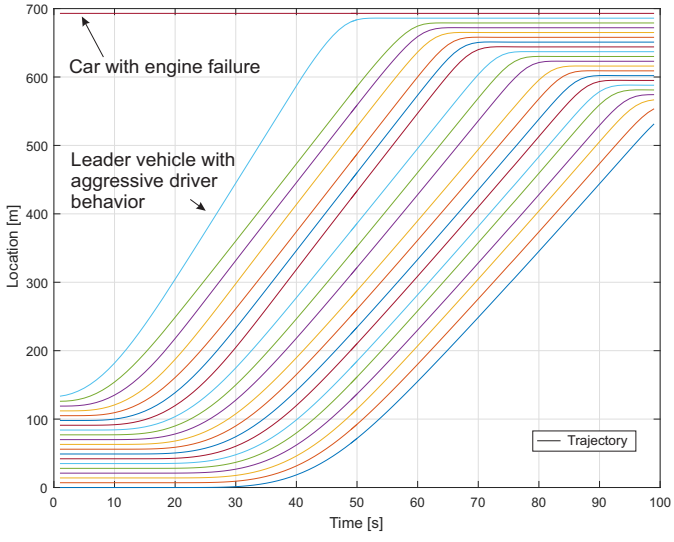


Fig. 7. Space-Time diagram for an inner-city scenario: the vehicles start in a jam situation, move into free traffic and are stopped by an obstacle in about 700m.

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 and "European Regional Development Fund" (EFRE) 2014-2020 in the course of the "InVerSiV"project under grant number EFRE-0800422 and has been conducted within the AutoMat (Automotive Big Data Marketplace for Innovative Cross-sectorial Vehicle Data Services) project, which received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement No 644657.

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.

[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.

[3] A. Virdis, G. Stea, and G. Nardini, *Simulating LTE/LTE-Advanced networks with SimuLTE*. Cham: Springer International Publishing, 2015, pp. 83–105.

[4] C. Sommer, R. German, and F. Dressler, "Bidirectionally coupled network and road traffic simulation for improved IVC analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, Jan 2011.

[5] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - simulation of urban mobility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, December 2012.

[6] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Phys. Rev. E*, vol. 62, pp. 1805–1824, Aug 2000.

[7] A. Kesting, M. Treiber, and D. Helbing, "General lane-changing model MOBIL for car-following models," *Transportation Research Record*, pp. 86–94, 2007.

[8] J. Pillmann, B. Sliwa, J. Schmutzler, C. Ide, and C. Wietfeld, "Car-to-cloud communication traffic analysis based on the common vehicle information model," in *IEEE Vehicular Technology Conference (VTC-Spring) Workshop on Wireless Access Technologies and Architectures for Internet of Things (IoT) Applications*, Jun 2017.

[9] M. Treiber and A. Kesting, "Traffic flow dynamics," *Traffic Flow Dynamics: Data, Models and Simulation, Springer-Verlag Berlin Heidelberg*, 2013.

[10] A. Mei and J. Stefa, "SWIM: A simple model to generate small mobile worlds," in *IEEE INFOCOM 2009 - The 28th Conference on Computer Communications*. IEEE, Apr. 2009, pp. 2106–2113.

[11] A. Udugama, B. Khalilov, A. bin Muslim, A. Foerster, and K. Kuladinithi, "Implementation of the SWIM mobility model in OMNeT++," in *OMNeT++ Community Summit 2016*, September 2016.

Observations on OMNeT++ Real-Time Behaviour

Christina Obermaier, Christian Facchi
Research Centre, Technische Hochschule Ingolstadt
Email: {christina.obermaier, christian.facchi}@thi.de

***Abstract*—*OMNeT++* is a widely used platform for all types of network simulations. The open source simulation framework *Artery* can be used to perform Vehicular Ad Hoc Network (VANET) simulations. This paper presents an approach for connecting this simulation and real-world VANET hardware to extend the test range and investigates the real-time behaviour of the simulation. As a Device Under Test (DUT) depends on real-time data to perform properly, different simulation scenarios running different hardware setups are presented. Additionally, the paper deals with the impacts of real-time losses on the test run outcomes. Most time dependant algorithms like the duplicate packet detection do not need very accurate real-time data and thus could be verified using the presented approach. Otherwise, in some cases such as testing of multi-hop communication, accurate real time is crucial.**

***Index Terms*—Vehicular Ad Hoc Network, Simulation, Hardware in the Loop**

I. Introduction

In times of increasing complexity of advanced driver assistance systems, it is crucial to enhance the environmental awareness of vehicles. Vehicles can be equipped with Vehicular Ad Hoc Network (VANET) devices, acting as a new information source besides already well known sensors.

VANETs are spontaneously created networks between road participants and Road Side Units (RSUs). They are based on IEEE 802.11p [1] and ETSI ITS G5 [2] in Europe as well as IEEE WAVE in the USA [3]. This paper focuses on the European ETSI ITS G5 standards.

VANETs and their applications were developed with the focus on enhancing traffic safety and traffic flow [4]. Especially in critical driving situations, availability of information is crucial. This leads to the question, how to test VANET communication properly. As Software in the Loop (SIL) simulation is not enough to ensure the availability, this paper evaluates if *Artery* can fulfil the real-time requirements necessary for Hardware in the Loop (HIL) testing.

In section II a overview of the related testing frameworks and hardware testbeds is given. Section III presents the state of the art and the theoretic concept of the hardware testbed. In section IV different scenarios running on different hardware setups are investigated. Section V presents the limitations of the presented approach. Finally, section VI includes a conclusion and a brief outlook.

II. Related Work

Currently there are mainly two different Inter-Vehicular-Communication (IVC) testing approaches. On the one hand, there is pure software testing with frameworks like *Artery*[1] [5] and *Veins*[2] [6] which are based on the discrete event simulator *OMNeT++*. These implement the Intelligent Transport System (ITS) G5 and the Wireless Access in Vehicular Environments (WAVE) standards, respectively. Also, different environmental circumstances and accidents can be modelled easily [7].

On the other hand, there are real-world testing approaches like the "Testfeld A9" established near Ingolstadt in Germany. Real-world testing allows for testing actual IVC hardware, but it is required to have at least two drivers in real vehicles equipped with IVC hardware. Thus, field tests are hard to reproduce and very expensive. While it is still possible to perform simple real-world scenarios, like presented in [8], it might be not feasible to do this with more complicated scenarios.

HIL tests can be used to perform hardware tests which are easy to repeat and cost effective. It is not a new approach using *OMNeT++* to connect a simulation with real-world hardware as *OMNeT++* was one of the fastest simulators in the domain of wireless networks in prior software versions [9]. In [10], a routing framework for *OMNeT++* HIL simulations is presented and the real-time behaviour of *OMNeT++* was investigated in different scenarios. They mentioned that real time will be a problem in future scenarios, especially if the node topology begins to change dynamically due to node mobility. According to [11], *OMNeT++* can also be coupled with *RoSeNet* to connect with real-hardware sensors. In [12] the performance of the *INET* framework (release 2.5.1) in emulation mode combined with an enhanced real-time scheduler was investigated. They pointed out to have performance problems. Additionally, they observed packet losses occurring in the communication between the real hardware and the simulation.

III. HiL concept

Artery is an open source framework for the discrete event simulation *OMNeT++* [5]. It allows for simulating European VANETs using *Vanetza*[3], which is an open source implementation of the ETSI ITS G5 communication stack [13]. *Veins* or *INET* provide the physical and Medium Access Control (MAC) layers. Moreover, the movement of the vehicles is simulated by the open source traffic simulator *SUMO*[4]. *SUMO* and *Artery* are coupled using the Traffic Command Interface (TraCI) [14].

[1] https://github.com/riebl/artery
[2] http://veins.car2x.org
[3] http://vanetza.org/
[4] http://sumo.dlr.de/

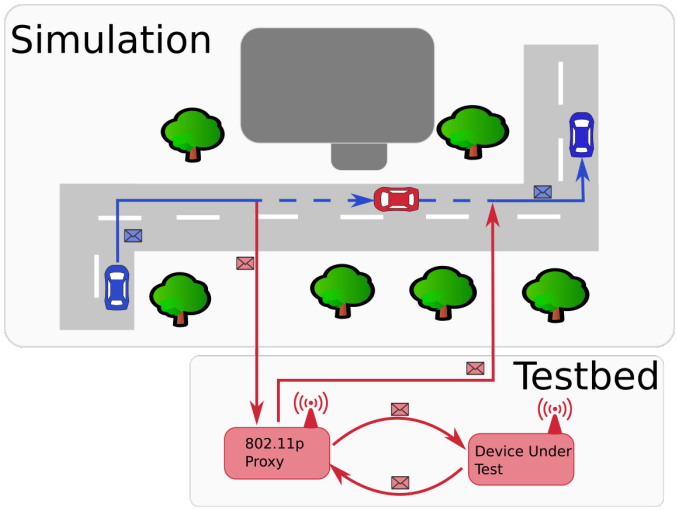


Figure 1. HiL concept overview

A. Artery Overview

Basically, each vehicle controlled by *SUMO* is represented by an *OMNeT++* compound-module called *Car*. A *Car* module is, among others, composed of a *Middleware*, a *VanetNic* and a *Mobility* submodule. The *Mobility* module is responsible for all vehicle dynamics related data and information. The *VanetNic* represents the network interface of each car. The *Middleware* module hosts all registered applications and contains an instance of *Vanetza*, taking care of routing and transport the incoming and outgoing IVC packages. Currently, the day one applications Cooperative Awareness Message (CAM) and Decentralized Environmental Notification Message (DENM) are provided by *Artery* [5], [15].

Cooperative Awareness (CA) is responsible to inform all road users in the network about basic data of all other participating vehicles. This basic data contains, among others, the network node's position, speed and heading. CAMs are generated up to ten times per second from each road participant [16]. In contrast, DENMs are only generated in case of a specific event. A DENM contains information of hazards and dangerous situtations occurring in road transport [17]. Thus, *Artery* allows for defining this dangerous situations and environmental influences like, for example, heavy rain or nearly accidents [7]. These situations are required to trigger related DENM messages.

B. HiL concept

Figure 1 shows the basic concept of a HIL simulation using *Artery* to provide test data. The red coloured vehicle is the simulated representative of a Device Under Test (DUT). For now, this car will further be called physical twin. If a message is going to be transmitted to the physical twin, the message will be transferred to a 802.11p gateway realised on software defined radio technology. This behaviour is represented by red arrows, showing the connection of the simulation to the real word and thus with the DUT. The blue dotted arrows show the current behaviour of the simulation.

Table I
HARDWARE SETUP

Component	Laptop Computer	Simulation Cluster
CPU	Intel Core i5-6300U @ 2.40GHz	Intel Xeon E7-8867 v4 @ 2.40GHz
Cores	1 x 4	4 x 18
RAM	16GB	3TB
Hard Drive	256GB SSD	450GB SAS SSD RAID 1

In contrast to other simulated cars, the physical twin has a quite limited functionality: Upper protocol layer processing is done by the DUT so the *Middleware* module as well as the *Vanetza* instance can be dropped. Otherwise, the *Mobility* module is still needed to provide Global Positioning System (GPS) data to the DUT to ensure properly working routing protocols [18]. Without appropriate GPS information, the geographic routing protocols defined in [18] would not work properly. Summarising, all functionality beginning with the MAC layer processing is stripped from the physical twin. It is only responsible for calculating message receptions and feeding received messages back on the *OMNeT++* channel.

As each kind of hardware testbed is dependent on real-time execution, the simulated environment must run in real time, too. To ensure this, the *OMNeT++* built-in scheduler is exchanged by a real-time scheduler. This scheduler is built upon *Boost ASIO* timers, ensuring asynchronous waiting. Hence, the real-time scheduler slows down the simulation, if it could run faster than real time. Additionally, it is aware of real-time losses so that it could stop the simulation if data could not be provided in real time. Also, the scheduler provides logging mechanisms to investigate the real-time behaviour of simulation runs after they are finished. A pseudocode implementation of the scheduler can be found in Appendix A.

IV. Investigations on Real-Time Behaviour

As already known from investigations described in [19] there is a quite low execution speed of *OMNeT++* VANET simulations. This leads to the question, if *Artery* is capable of reaching and holding real time to provide data for a HIL testbed while doing an online simulation.

A. Scenario Description

The chosen scenario is very simple: Three vehicles driving on a highway from north to south.

As the amount of driving vehicles seems to be the main influence on the execution speed of these simulations, the second test scenario increases the amount of driving vehicles by two. All other parameters remain the same as in the previous scenario. The setup of the radio medium is configured by *Artery* using its *INET* defaults. The in Section III-B introduced scheduler is used to evaluate the real-time behaviour. The simulation was built using *OMNeT++* version 5.1.1 and was executed on two different computers: A laptop computer and a simulation cluster. Table I presents a comrehensive hardware list of the used computers. As *OMNeT++* uses only one core

Table II
EVENT MAPPING

ID	Event name	# Events "3 vehicles"	# Events "5 vehicles"
1	TraCI Connect	1	1
2	TraCI Step	322	370
3	GeoNet packet	3870	11298
4	GeoNet data frame	3870	11298
5	txStart-0	3	5
6	endIFS	661	1189
7	configureRadioMode	1322	2378
8	transmissionTimer	661	1189
9	remove non Interfering Transmission	661	1188
10	report CL	928	1650
11	middleware update	925	1645
12	txStart-1	658	1184
13	GeoNet radio frame	1274	4460
14	reception Timer	1274	4460
	Overall events	16430	42315

in this test setup, the speed-up of the simulation running on the simulation cluster is caused by faster calculation hardware not by better parallelism.

B. Simulation Results and Scenario Comparison

Table II presents the number of events occurring in both scenarios. The ID column is used to map event IDs to the events depicted in the box plots in figure 2a and 3a. The third and the fourth columns show the number of occurring events dependent on the simulated scenario. Thus, compared to the base scenario which triggers 16430 events, the scenario with two more vehicles triggers 42315 events. Hence, adding two cars causes 2.58 times more events in this scenario setup.

Figure 2 and 3 present the evaluations of both scenarios executed on the simulation cluster and the laptop computer. The included boxplots depict the execution time of events. The indices on the x-axis correspond to the events mentioned in Table II. If we compare both figures, the time a event needs to be executed is lower while using the simulation cluster but relative event durations remain nearly the same. This behaviour is caused by the faster computing capabilities of the single cores of the simulation cluster.

The histograms 3b and 3c show the real-time misses per event while the scenarios were executed on the laptop. Histogram 2b and 2c present the same but for the execution on the simulation cluster. As the green bars indicate events executed nearly in real time, higher green bars and lower red bars indicate a nearly real-time capable simulation run.

It can be seen that there are real-time drops up to 1.5 seconds in each scenario. This is caused by the *TraCi Connect* event, which is executed as first event and takes about one second. Thus, even if the scenario itself is real-time capable, there will be always a few seconds at the beginning of the simulation which must be skipped because of the system startup. Figure 2b is a great example for a good and fast running scenario. There are only roughly 50 - 100 events which are more than one second behind the real time. About 15000 events out of the total amount of 16430 events are executed nearly in real time. This means, the simulation cluster can basically handle a small scenario with three vehicles in real time.

Figure 3b presents the same scenario executed on a laptop computer. It can be seen that there are still about 10000 events executed nearly in real time. Other 5000 events are executed only 0.1 second behind real time. Thus, only a few more events can be found in the area of about one second behind real time, but the scenario looks still quite good.

Distinct differences between execution speed of the computing hardware can be seen in Figure 2c and Figure 3c. The simulation cluster is able to handle the five car scenario with still up to 1.5 seconds real-time loss. The amount of events in the area from 1 to 1.5 seconds is significantly increased, compared to the base scenario but *OMNeT++* is still able to catch up. However, the five car scenario executed on the laptop computer exceeds the 1.5 seconds limit by far. There are many real-time losses up to three seconds. Thus, the five car scenario can only be handled by the simulation cluster.

Figure 4 admits a closer look on the real-time behaviour of the different simulation runs. Blue bars show the time needed to process all events occurring at a particular simulation time stamp. The red line depict the current gap between simulation time and real time. Thus, a higher blue bar causes a higher real-time loss peak. It seems that *Artery* tends to schedule a bunch of events every 50ms. This behaviour causes the simulation to fall behind real time every 50ms, which is indicated by the peaks of the red line.

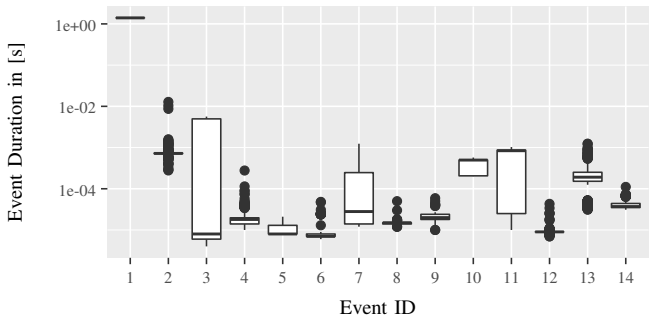
As it can be seen, the simulation cluster produces as many real-time losses as the laptop computer. But, even if the differences between the execution time of single events are not that clear, the real-time losses are much higher on the laptop computer, independent from the executed scenario. This approves that *Artery* has to execute many events at the same time because they are scheduled at nearly the same timestamp. Due to the lack of parallelism, *OMNeT++* has to execute this events in a sequence, which causes the real-time losses.

As there are many events which are closely linked together, for example the *GeoNet packet* and the *GeoNet data frame*, this behaviour could not be changed significantly. Thus, there will always be real-time losses caused by many events occurring at the same time if there is no parallelism.

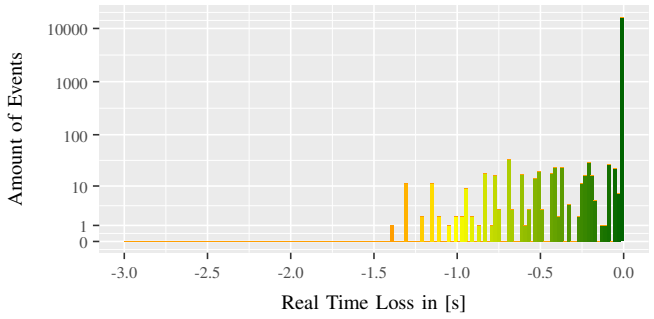
C. OMNeT++ Time Flow

Figure 5 depicts a generic example how the timeline in *OMNeT++* behaves compared to the real-time flow.

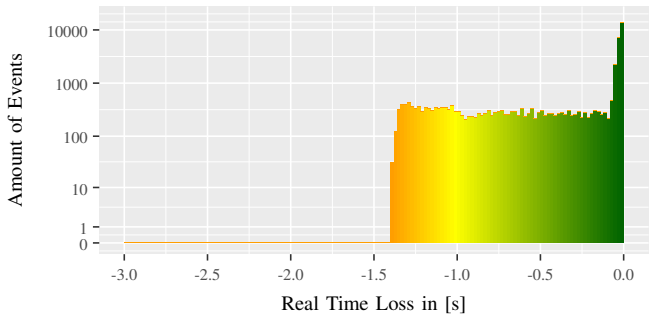
As it can be seen, the absolute time needed to execute one simulation millisecond is sometimes higher than one real millisecond like indicated at millisecond two. If the simulation is behind the wall-clock time, it has to catch up to avoid higher real-time gaps. Hence, the simulation is trying to execute



(a) Average event times

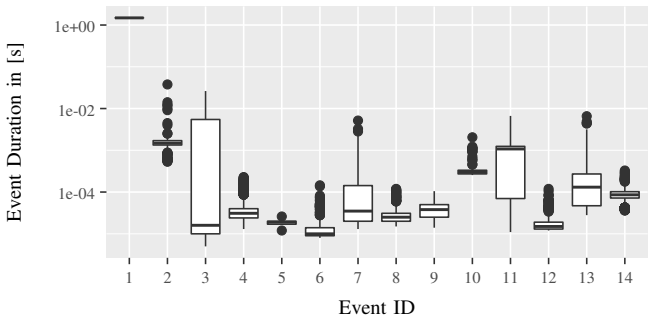


(b) Amount of critical real-time losses in base scenario

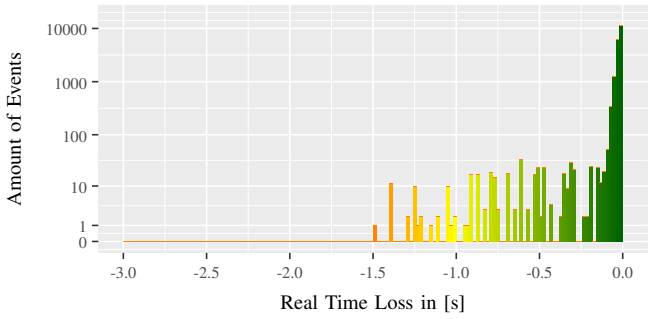


(c) Amount of critical real-time losses in five cars scenario

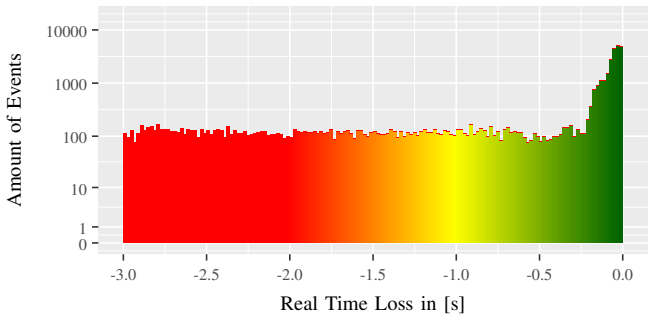
Figure 2. Scenarios executed on the simulation cluster



(a) Average event times



(b) Amount of critical real-time losses in base scenario



(c) Amount of critical real-time losses in five cars scenario

Figure 3. Scenario executed on the laptop computer

events faster than in real-time till simulation time and wall-clock time are matching again. But, even if the *OMNeT++* clock sometimes progresses faster than real time, the fixed time stamp is always behind or equal to the wall-clock time which is ensured by the real-time scheduler. If the next event is located in the future, the scheduler waits till the timestamp of the event matches the wall-clock time. Thus, it is ensured that the DUT must only deal with timestamps in the past and not in the future, which would be much more problematic.

V. Simulation Result Interpretation

Most scenarios in IVC communication do not depend on very accurate timestamps. For example, the Long Position Vector (LPV) [18, Section 8.8] contains the time at which the last geographic position update was received by the vehicle writing the LPV. As in a real-world environment GPS update rates are fluctuating, the timestamp in the LPV is not strongly related to the current wall-clock time.

The LPV timestamp is used to perform duplicate packet detection at the network layer [18, p. 63]. The duplicate packet detection algorithms depend on the sequence number as well as the timestamp. A packet is identified to be a duplicate if the timestamp is lower or equal to an already received packet. As the introduced *RealTimeScheduler* ensures that the simulation time is always behind or equal to the wall-clock time, this algorithm works like expected.

CAMs contain a generation timestamp [16] used, for example, to recognise and avoid replay attacks [2, Section 7.6.1]. However, a CAM is not detected to be a duplicate if there is only a slight difference between wall clock time and the generation timestamp.

A DENM contains various timestamps as well. There is, for example, an expiry time after which a DENM event is terminated. Mostly, this period lasts several seconds but in cases like the dangerous situation [20] trigger, a DENM event lasts only two seconds. Thus, in some situations a real-time

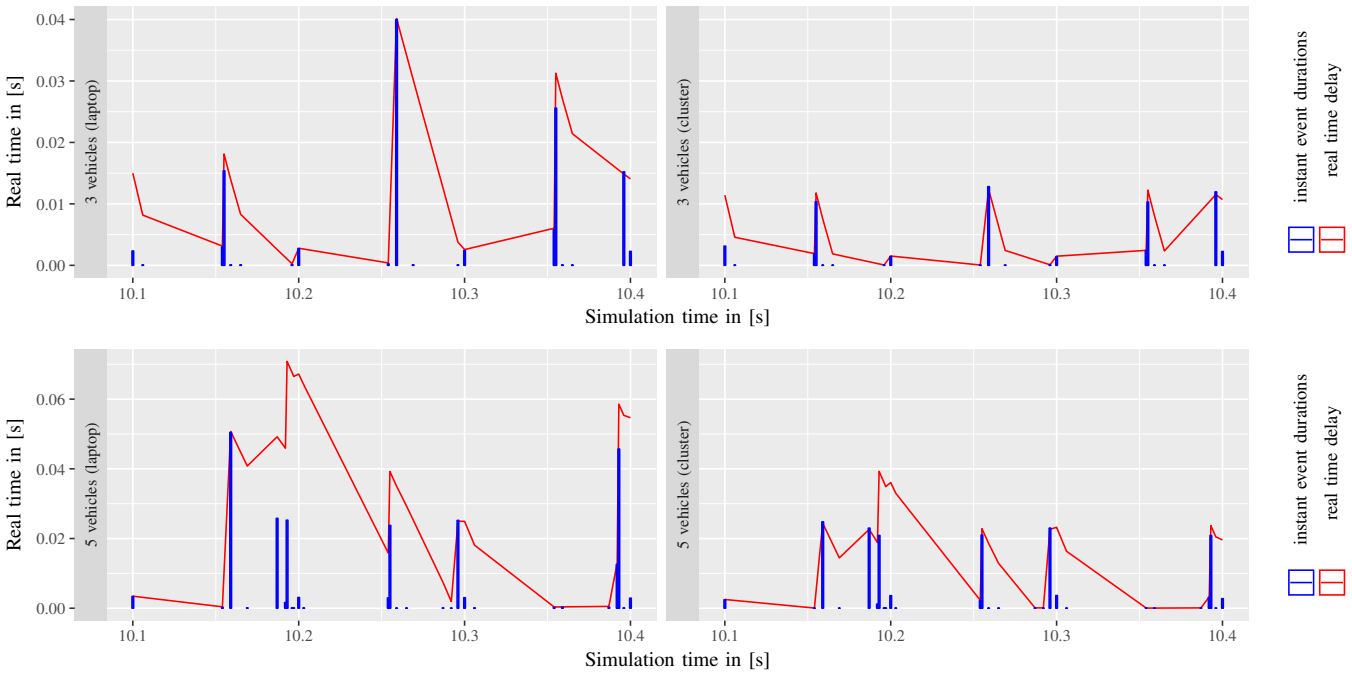


Figure 4. Real-time delays and event durations

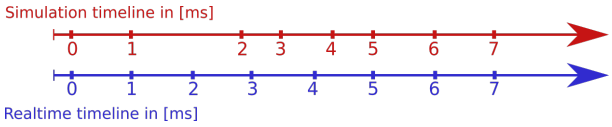


Figure 5. OMNeT++ and real-time timeline

loss higher than one ore two seconds may invalidate a DENM wrongly.

Another issue is the long-range communication using multiple vehicles as forwarders [18]. A MAC layer unicast is followed by an Acknowledgement Frame (ACK), sent by the receiver, to confirm a successful packet transmission. The time span in which the ACK is expected by the initial sender is the time of a Short Interframe Space (SIFS) [1, Section 9.3.2.8]. Determined by the channel bandwith of 10 MHz [21] the SIFS is 32 μs [1, Table 18-17]. This very short time span leads to a potential problem: If a MAC layer unicast is sent to the DUT, it has to respond with an ACK within this SIFS. As depicted in Figure 5 between millisecond two and three, *OMNeT++* could run faster than real time. Thus, the sending vehicle inside *OMNeT++* may not receive this message in the claimed timespan. Hence, this problem must be solved to perform multi-hop tests.

Summarizing, in most cases, a real-time loss in a range of a few milliseconds is not that problematic. Important algorithms like the duplicate packet detection do only rely on linear time flow and do not depend on hard real time. Only a few scenarios like multi-hop testing relay on a very accurate time synchronisation. Thus, *Artery* is basically able to provide online simulation data for a real-time testbed.

VI. Conclusion

This paper presented an approach for extending *Artery* to provide hardware tests. *Artery* is used to facilitate SIL tests in the area of VANETs. As a HIL simulation always depends on real-time data, the online simulated scenarios have to be executed in this manner. It was investigated that this criteria can only be fulfilled if only a few cars are simulated. Also, even if the simulation is overall real-time capable, there are always real-time losses. How significant these losses are depends on the used computation hardware and the scenario complexity. As *OMNeT++* only uses one core when simulating VANETs, a higher single core performance causes a higher execution speed. Also, *Artery* and other wireless communication models tends to produce events to be executed nearly at the same time, causing more significant real-time losses. This is related the fact that one sending event triggers various receiving events.

Also, it was audited in which situations nearly hard real time is required and when real-time drops are bearable: Multi-hop communication strongly depends on real time data, so its not possible to test this feature properly in the current state of the simulation. Most other algorithms do only depend on a steady time flow and are not influenced by real-time losses in the range of a few milliseconds. This is the case for DENM message expiries or the recognition of replay attacks. Also the duplicate packet detection is not affected harmfully.

In conclusion, *Artery* could basically be used to provide data for VANET HIL tests. In case of multi-hop-test scenarios, real-time losses may influence the test results heavily. Other scenarios can be used to provide functional testing of VANET hardware. This scenarios are currently limited to three or four

vehicles simulated vehicles depending on the used hardware.

As this paper presents a work in progress research project, the HIL testbed will be created with the observed behaviour of *OMNeT++* in mind. Thus, to enable multi-hop testing, which is crucial in European VANETs, one idea is to use a Software Defined Radio (SDR) as 802.11p proxy. This allows for a modified MAC layer of the used proxy device to send ACK frames depending on the current state of the simulation. Hence, the proxy device must know all vehicles which can communicate with the DUT in the current simulation step. This idea could provide a basic implementation of a testbed which can handle most functionality of ETSI ITS G5 networks with the constraint that the simulation must run in nearly real time.

So, further work must be done in the field of enhancing simulation speed to achieve faster running simulation scenarios. Moreover, other ways to provide test data for hardware tests can be investigated. This includes, among others, the capturing of the simulated network traffic and playing them back in real time.

Appendix A
Real Time Scheduler

Result: next cEvent
currentRealTimeMiss = *simTime* - *wallClockTime*;
if *(currentRealTimeMiss * -1) > realTimeMissThreshold* **then**
    `// simulation unacceptable slow`
    stop simulation;
else
    eventDuration = *wallClockTime* - *eventStartTime*;
    log *currentRealTimeMiss* and *eventDuration* and *nextEventIdentifier*;
    while *SimTime > wallClockTime* **do**
        `// simulation faster than real time`
        wait;
    end
    set *nextEventIdentifier*;
    set *eventStartTime*;
    return *nextEvent*;
end

Algorithm 1: *cEvent* RealTimeScheduler::takeNextEvent* pseudocode

References

[1] *IEEE 802.11 Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, Institute of Electrical and Electronics Engineers, Mar. 2012.

[2] *EN 302 665 Intelligent Transport Systems (ITS); Communications Architecture*, European Telecommunications Standards Institute, Sep. 2010.

[3] *IEEE 1609.0-2013 IEEE Guide for Wireless Access in Vehicular Environments (WAVE) - Architecture*, 2014.

[4] T. L. Willke, P. Tientrakool, and N. F. Maxemchuk, "A survey of inter-vehicle communication protocols and their applications", *IEEE Communications Surveys Tutorials*, vol. 11, no. 2, pp. 3–20, Second 2009.

[5] R. Riebl, H. J. Günther, C. Facchi, and L. Wolf, "Artery: Extending Veins for VANET applications", in *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, Jun. 2015, pp. 450–456.

[6] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis", *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, Jan. 2011.

[7] C. Obermaier, R. Riebl, and C. Facchi, "Dynamic Scenario Control for VANET Simulations", in *Proceedings of IEEE MT-ITS 2017*, Unpublished, 2017.

[8] K. C. Lee, S. h. Lee, R. Cheung, U. Lee, and M. Gerla, "First Experience with CarTorrent in a Real Vehicular Ad Hoc Network Testbed", in *2007 Mobile Networking for Vehicular Environments*, May 2007, pp. 109–114.

[9] A. R. Khan, S. M. Bilal, and M. Othman, "A performance comparison of open source network simulators for wireless networks", in *2012 IEEE International Conference on Control System, Computing and Engineering*, Nov. 2012, pp. 34–38.

[10] G. Nirav and K. M. Sivalingam, "Dynamic routing framework for OMNeT++ based Hardware-In-The-Loop (HITL) network simulation", in *2014 Twentieth National Conference on Communications (NCC)*, Feb. 2014, pp. 1–6.

[11] S. Boehm and M. Kirsche, "Looking into Hardware-in-the-Loop Coupling of OMNeT++ and RoSeNet", in *Proceedings of the "OMNeT++" Community Summit 2015*, 2015.

[12] A. A. Scussel, G. Panholzer, C. Brandauer, and F. von Tüllenburg, "Improvements in OMNeT++/INET Real-Time Scheduler for Emulation Mode", in *Proceedings of the OMNeT++ Community Summit 2015*, 2015.

[13] R. Riebl, C. Obermaier, S. Neumeier, and C. Facchi, "Vanetza: Boosting Research on Inter-Vehicle Communication", in *Proceedings of the 5th GI/ITG KuVS Fachgespräch Inter-Vehicle Communication (FG-IVC 2017)*, Apr. 2017, pp. 37–40.

[14] A. Wegener, M. Piórkowski, M. Raya, *et al.*, "TraCI", in *Proceedings of the 11th communications and networking simulation symposium on - CNS '08*, ACM Press, 2008.

[15] R. Riebl. (2017). Artery, [Online]. Available: https://github.com/riebl/artery (visited on 05/13/2017).

[16] *EN 302 637-2 Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service*, European Telecommunications Standards Institute, Nov. 2014.

[17] *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service*, European Telecommunications Standards Institute, Nov. 2014.

[18] *EN 302 636-4-1 Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 1: Media-Independent Functionality*, European Telecommunications Standards Institute, Jul. 2014.

[19] R. Riebl, S. Neumeier, and C. Facchi, "Inter-Vehicle Communication on the Run - Experiences From Tweaking Veins Runtime Performance", Ulmer Informatik-Berichte, 2015.

[20] T. Biehle and K. Krumbiegel, *Triggering Conditions and Data Quality - Dangerous Situation*, Release 1.1.0, CAR 2 CAR Communication Consortium, 2015.

[21] *EN 302 663 Intelligent Transport Systems (ITS); Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band*, European Telecommunications Standards Institute, Jul. 2013.

Simulating cellular communications in vehicular networks: making SimuLTE interoperable with Veins

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

***Abstract*— The evolution of cellular technologies toward 5G progressively enables efficient and ubiquitous communications in an increasing number of fields. Among these, vehicular networks are being considered as one of the most promising and challenging applications, requiring support for communications in high-speed mobility and delay-constrained information exchange in proximity. In this context, simulation frameworks under the OMNeT++ umbrella are already available: SimuLTE and Veins for cellular and vehicular systems, respectively. In this paper, we describe the modifications that make SimuLTE interoperable with Veins and INET, which leverage the OMNeT++ paradigm, and allow us to achieve our goal without any modification to either of the latter two. We discuss the limitations of the previous solution, namely VeinsLTE, which integrates all three in a single framework, thus preventing independent evolution and upgrades of each building block.**

Keywords*— *SimuLTE, Veins, vehicular networks, cellular networks, LTE, 5G

I. Introduction

Over the last two decades, two independent research areas have witnessed huge progress: cellular communications, on one side, and vehicular networks, on the other. Parallel to increasing the end-user bandwidth and providing ubiquitous coverage, research on cellular networks has also defined and analyzed a growing number of use cases, incorporating their requirements in the latest standards. This is especially evident with the onset of 5G research, a key issue of which is providing guaranteed performance to the largest possible number of users, regardless of their position. This requires the network to be deployed in a denser manner, to support high-speed mobility in a reliable manner, to perform fast handover between neighboring cells, etc.. Parallel to this, and quite likely due to the above improvements, research on vehicular technologies has started to consider cellular networks as a promising communication technology. On one hand, cellular communications allow existing vehicular network services to be improved, and new ones to be enabled. On the other hand, recent research projects envision vehicular communications as a promising use case for cellular systems, under the name of "connected cars" [1], [2].

The interactions between the above two fields can be seen from two main points of view: first, it can enable the so-called vehicle-to-everything (V2X) communications, including communications among vehicles (V2V) or between cars and the infrastructure (V2I), i.e. scenarios where vehicles are either or both end-points of the communication. Second, vehicular mobility can be seen as a specific case of user mobility, thus introducing new challenges to the underlying communication network, such as bulk handovers, unexpected network congestions during traffic jams, etc.

Network simulation has been widely used in both fields to test network architectures, algorithms, communication protocols, and applications on a large scale. In the frame of OMNeT++, there are two frameworks that tackle vehicular networks and cellular systems, called Veins [3] and SimuLTE [4][5], respectively. Veins provides vehicular mobility to OMNeT++, using *SUMO* [6] as the underlying vehicular-traffic simulator. SimuLTE, instead, is a system-level-simulator of LTE networks, based on the INET framework. Although the two frameworks are developed for the same simulation system, they cannot be used together "as-is". A first integration attempt has been made with VeinsLTE [7], which integrates a customized version of both in a single package. However, this solution defines a third standalone framework, rather than interconnecting the two, taking snapshots of two independent ongoing developments. This makes it difficult, if possible at all, to upgrade it when the composing framework gets upgraded. For instance, the recent additions of device-to-device communications in SimuLTE [8],[9] are not automatically subsumed in VeinsLTE. Dually, the addition of the support to platooning [10] in Veins does not automatically enable it in VeinsLTE.

In this paper, we describe the process of making SimuLTE and Veins *interoperable*, i.e. using both in the same simulation scenarios with the specific aim of keeping them separate and independent. We highlight the requirements coming from Veins and we list the main modifications to SimuLTE required to satisfy these requirements, in particular detailing how we manage dynamic creation and destruction of LTE-capable nodes. The rest of this paper is organized as follows: Section II provides the background on SimuLTE. In Section III we describe the requirements coming from Veins and the related work on the topic. Section IV, then, describes the integration process and Section V provides an example of simulation configuration. Finally, Section VI concludes the paper.

II. Background on SimuLTE

In this section, we provide a background on the main architectural elements of SimuLTE. The latter is a framework

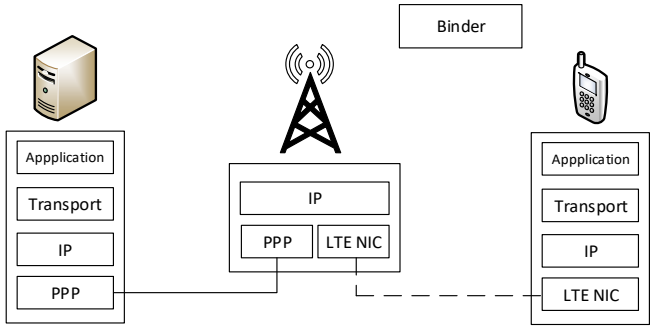


Figure 1 - High-level representation of an LTE scenario using SimuLTE

for system-level simulation of LTE and LTE-Advanced cellular networks. It is based on OMNeT++ and exploits the INET framework to implement all the higher layers of the IP stack as well as the main IP nodes of the communication network, such as IP routers and application servers. Moreover, INET provides the concept of Network Interface Card (NIC) modules, which can be included within other modules to implement models of various communication protocols between network devices. As an example, devices can be empowered with Wi-Fi or Point-to-Point Protocol (PPP) communication capabilities by integrating the respective NIC modules. SimuLTE exploits the concept of modularity coming from OMNeT++ to realize its main building block: the LTE NIC card. The latter is designed as an extension of a wireless NIC module from INET, and allows one to add LTE capabilities to a node included in the simulation.

SimuLTE provides models for the two main elements of the LTE communication architecture, namely the user equipment (UE) and the evolved NodeB (eNB). Both nodes contain the LTE NIC, as shown in Figure 1, together with modules implementing upper layer protocols, taken from INET. The eNB has also an interface to the Internet via PPP, which provides means to send/receive data traffic coming from application servers, implemented as INET *standard hosts*. Multiple eNBs can also be connected together using the X2 interface, enabling coordination algorithms such as CoMP [11]. Within the LTE NIC located at both the UE and the eNB, SimuLTE implements a complete LTE protocol stack by means of a submodule for each layer, namely Packet Data Convergence Protocol (PDCP), Radio Link Control (RLC), MAC and PHY. SimuLTE exploits the inheritance paradigm of OMNeT++ to define both structure and behavior of UEs and eNBs. According to this paradigm, each layer will have common operations implemented in a base class. The latter is then extended to implement functionalities specific for the UE and the eNB, respectively. Both communication between adjacent layers and data transmission between UEs and eNBs are implemented via message exchange. Accounting of transmission resources is managed separately from data transmission, i.e. transmission signals are not explicitly modeled. To do so, the *Binder* module keeps track of how many and which transmission resources, i.e. Resource Blocks (RBs), are used during each transmission slot of 1ms. Two distinct sets of RBs are used respectively by the eNBs for downlink transmissions, and by the UEs for uplink transmissions. The Binder thus acts as an oracle for the LTE network: all the LTE nodes in the simulated network register to it at the beginning of the simulation, and can access it via direct method calls to share or obtain common information. For instance, this allows us to model control-plane operations with negligible cost in term of complexity.

The ChannelModel class included within the PHY layer of each LTE NIC models the effects of the air channel on transmissions between nodes. Whenever a new message reaches a NIC, the corresponding ChannelModel evaluates the Signal-to-Interference-and-Noise Ratio (SINR) as perceived by the node. This computation accounts for the intercell interference, which affects LTE communications. To do so, the channel model on the receiving end obtains information from the Binder regarding the RB allocation for all the nodes in the network. The compute SINR is then used to decide whether or not the message can be successfully decoded, depending on its value and on the modulation and coding scheme (MCS) used for that transmission. The ChannelModel is also responsible for the computation of the UE's Channel Quality Indicator (CQI). The latter is used by the eNB to select the MCS used for transmission, and might be also used for scheduling operations at the eNB, e.g. to enforce maximum-throughput allocation schemes. SimuLTE provides a realistic implementation of the ChannelModel, which takes into account path loss and fading effects. Such implementation can be easily extended or replaced, thus allowing researchers to use whatever physical-layer model they see fit.

SimuLTE allows one to simulate both communication between UEs and eNBs, and among UEs. If two UEs are both served by the same eNB, they can communicate either using the latter as a relay or in a direct manner, called *device-to-device* (D2D) communication [12]. Moreover, the LTE NIC provided by SimuLTE can be used to empower any network device with LTE communication capabilities. This way it is possible to design nodes with multiple and heterogeneous network interfaces, e.g. LTE, Wi-Fi, Bluetooth, etc.

III. VEINS REQUIREMENTS AND RELATED WORK

In this section, we describe the requirements that SimuLTE must meet in order to integrate vehicular mobility provided by the Veins framework. Then, we give an insight on the limitation of the existing solution, namely *VeinsLTE* [7].

Support to vehicular mobility within Veins is provided by *SUMO* [6], a popular simulation tool that allows users to create detailed road traffic scenarios. The responsibility of interacting with SUMO through the *TraCI* interface [8] is given to a module called *TraCIScenarioManager*, which must be present in any network definition. In particular, the TraCIScenarioManager obtains the information about the movements of vehicles in the simulated road traffic scenario and updates their mobility information within OMNeT++. To do this, vehicles (defined as OMNeT++ compound modules) need to be equipped with a *TraCIMobility* submodule. For vehicular mobility to work with SimuLTE, UEs need to incorporate this type of mobility. However, in SimuLTE, mobility of UEs is handled by the INET framework and its latest versions do not provide TraCIMobility. This means that modifications are required to add this support. Vehicles can also enter and/or leave the simulation dynamically, i.e. they can

appear/disappear at any time during the simulation. This is done by the `addModule` and `deleteManagedModule` functions implemented by the TraCIScenarioManager. The `addModule` function instantiates the new OMNeT++ module implementing the vehicle and calls its initialization procedure. The `deleteManagedModule`, instead, invokes the `finish` function of the module and removes it from the simulation. These operations are accomplished by using the OMNeT++ API. If the new module is an LTE-capable node, it needs to be registered to the Binder of SimuLTE, i.e. stored in the data structures of the Binder. Similarly, on deletion, it must be removed from the Binder's data structures. Moreover, when considering cellular communications, a newly created vehicle must be associated with its serving base station (i.e. the eNB in case of the LTE network). Typically, vehicular networks cover large geographical areas (e.g., a city), hence a *multicell* environment must be considered. Since the new vehicle can appear at any position of the simulation playground, a procedure that performs the attachment of the incoming vehicle to the best serving cell is necessary and this should occur during the initialization phase of the module. For example, the vehicle can be associated to the eNB from which it perceives the highest value of the Signal to Interference and Noise Ratio (SINR), although this is not the only possible association criterion. In any case, the best serving eNB can change during the simulation, due to mobility and/or varying conditions in the environment (e.g. interference). Thus, support for *handover* (HO) must be available too.

The first attempt to integrate SimuLTE and Veins has been made by *VeinsLTE* [7]. The latter provides a single package that puts together Veins, SimuLTE and INET. However, it does so by making modifications to both Veins and SimuLTE, such that they need to interact *directly* with each other. For example, registration to the Binder during the creation of new vehicles in the `addModule` function is performed via direct method calls. The same occurs when removing vehicles in the `deleteManagedModule` function. This is a problem if the data structures of the Binder are changed in future releases of SimuLTE, e.g. to add new functionalities. Moreover, the eNB to which the new vehicle has to be associated is defined statically by the `ini` file, with no possibility of selecting the best serving cell at runtime. As far as mobility is concerned, the INET version included in the VeinsLTE package (i.e. INET v2.3) provides the TraCIMobility modules, hence the only thing to do is to configure that mobility for LTE-capable nodes in the `ini` file. However, INET removed such support recently.

IV. Modifications to SimuLTE

In this section, we describe the main modifications to the code of SimuLTE to allow the interoperability with Veins.

Figure 2 shows the *Car* module implemented within SimuLTE, which is similar to the structure of a legacy UE. With respect to the latter, a *vehicularMobility* module has been added. This is defined as an interface that can be implemented by the TraCIMobility module defined by Veins. Note that the *INETMobility* module is still present, so as to maintain backward compatibility. Since the Car cannot use both

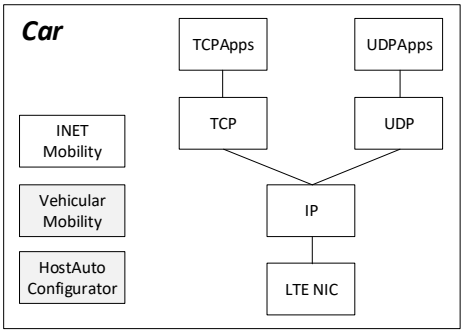


Figure 2 – Car module definition

mobility modules during the simulation, they are defined as conditional modules within the `ned` file. Anyway, both modules exploit the mechanism of *signals* provided by OMNeT++ to trigger mobility events, which are handled by modules that are *subscribed* to them. According to the activated mobility module, the Car subscribes itself to the signals generated by the corresponding module. Unlike with VeinsLTE, it is not necessary that the TraCIMobility module implements the INET mobility interface.

As far as dynamic creation of vehicles is concerned, Veins uses the OMNeT++ API to create and initialize the new module, as mentioned in Section III. However, previous releases of SimuLTE were designed so that modules' initialization was only performed at the beginning of the simulation. For this reason, we reorganized the initialization functions within the entire LTE protocol stack of the Car module, so that SimuLTE takes care of all the initialization related tasks of the LTE NIC, thus allowing the registration to the Binder at any time. Similarly, when a vehicle leaves the simulation, Veins just invokes the latter's `finish()` function, which in turn calls the `finish()` function for all the submodules of the vehicle to be removed, if implemented. Thus, we defined the behavior of those functions for the relevant submodules of the LTE NIC so as to deregister the vehicle from the Binder and clear the buffers (at both UE and eNB side).

When a new vehicle is created, it needs to obtain an IP address to communicate. SimuLTE demands the assignment of IP addresses to the *IPv4NetworkConfigurator* module provided by INET. However, this module performs the assignment only at the beginning of the simulation, hence it cannot handle modules created dynamically. To solve the problem, we endowed the Car with a *HostAutoConfigurator* module. The latter is a deprecated module that INET's authors have left with the specific purpose of enabling the addition of network nodes at runtime. New vehicles must also be associated to one eNB according to a given criterion. To do so, we allow the vehicle to measure the power received from every simulated eNB and to perform the attachment to the one with the largest value of received power, something that was not possible using previous SimuLTE versions. Mobility across different cells is handled by the existing HO mechanism provided by SimuLTE, which is transparent to the fact that the UE can be a vehicle. We remark that no modifications to Veins have been made to enable the above functionalities, making them independent

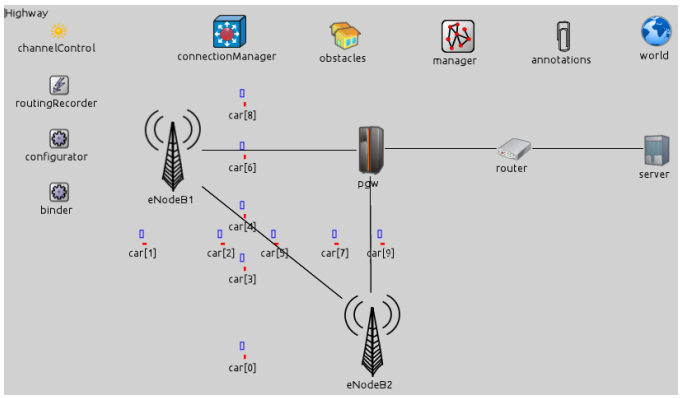


Figure 3 - Simulation scenario

from the employed version of the Veins framework, as long as the requirements described in Section III hold.

V. Simulating Vehicular Networks

In this section, we show how to configure a simulation that integrates SimuLTE and Veins in order to evaluate cellular communications in a vehicular environment. We describe only the (few) relevant parameters in the `ini` configuration file needed to accomplish this task, whereas we refer the reader to the exemplary configurations provided by SimuLTE and Veins frameworks to tune their respective parameters. Similarly, the configuration of the specific road traffic scenario is performed by SUMO and this is outside the scope of this paper.

An example of network definition is shown in Figure 3. It contains ten vehicles created dynamically and two eNBs connected via the X2 interface, and to a remote server through a simplified Evolved Packet Core (EPC) network. The definition of the network must comprise the Binder and the TraCIScenarioManager modules (*binder* and *manager* in Figure 3, respectively). The latter needs to be informed about the type of the vehicles it needs to add to the simulation. This is accomplished by setting the following parameters.

```
*.manager.moduleType="lte.corenetwork.nodes.cars.Car"
*.manager.moduleName="car"
```

The `moduleType` parameter defines the path of the `ned` definition of the Car module we described in Section IV. The `moduleName` parameter, instead, states that we can configure parameters related to the new modules by referring to them using the name `car`. For example, we can impose that the first added vehicle (namely, `car[0]`) encounters an accident after 20 seconds from its departure time and that the accident lasts 30 seconds. As the snippet of code below shows, this is done by modifying the parameters of the `vehicularMobility` module of `car[0]`.

```
*.car[0].vehicularMobility.accidentCount = 1
*.car[0].vehicularMobility.accidentStart = 20s
*.car[0].vehicularMobility.accidentDuration = 30s
```

Initial association of vehicles to the best serving eNBs is disabled by default, but it can be activated as follows:

```
**.dynamicCellAssociation = true
```

Otherwise, manual association must be specified as follows:

```
*.car[*].masterId = 1
*.car[*].macCellId = 1
```

In this example, all vehicles are associated to eNodeB1 during their initialization, regardless of their position. In any case, HO can be activated by specifying `**.enableHandover = true`.

VI. Conclusions and Future Work

In this paper, we described the integration of SimuLTE and Veins, with the specific goal of preserving them as independent frameworks. We first detailed the requirements coming from Veins to allow custom nodes to be managed according to its mobility model. We then described the modification we implemented on SimuLTE to fit the above requirements, thus creating scenarios where LTE capable nodes are moved according to vehicular patterns. Finally, we provided an example of network configuration, highlighting the main configuration parameters.

References

[1] Deliverable D1.1, Refined Scenarios and requirements, consolidated use cases, and qualitative techno-economic feasibility assessment, METIS-II Project, 31 Jan 2016

[2] Deliverable D1.1. 5G system use cases, scenarios, and requirements break-down, Flex5Gware 5G-PPP project, 31 Dec. 2015

[3] C. Sommer, R. German, F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis", IEEE Transactions on Mobile Computing, vol. 10 (1), pp. 3-15, January 2011.

[4] A. Virdis, G. Stea, G. Nardini, "Simulating LTE/LTE-Advanced Networks with SimuLTE", in M.S. Obaidat, J. Kacprzyk, T. Ören, J. Filipe, (eds.) "Simulation and Modeling Methodologies Technologies and Applications", vol. 402, pp. 83-105, 2015, Springer-Verlag.

[5] SimuLTE webpage. http://www.simulte.com.

[6] D. Krajzewicz, G. Hertkorn, C. Rössel, P. Wagner, "SUMO (Simulation of Urban MObility); An Open-Source Traffic Simulation", MESM 2002, pp. 183-187, September 2002.

[7] F. Hagenauer, F. Dressler, C. Sommer, "A Simulator for Heterogeneous Vehicular Networks", IEEE Vehicular Networking Conference (VNC) 2014, Paderborn, DE, Dec 2014.

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

[9] G. Nardini, A. Virdis, G. Stea, "Simulating device-to-device communications in OMNeT++ with SimuLTE: scenarios and configurations", OMNeT++ Community Summit 2016, Brno, CZ, September 15-16, 2016.

[10] M. Segata, S. Joerer, B. Bloessl, C. Sommer, F. Dressler, R. Lo Cigno, "PLEXE: A Platooning Extension for Veins", Proc. of IEEE VNC 2014, Paderborn, DE, December 2014.

[11] G. Nardini, A. Virdis, G. Stea, "Modeling X2 backhauling for LTE-Advanced and assessing its effect on CoMP Coordinated Scheduling", International Workshop on Link- and System Level Simulations (IWSLS2) 2016, Vienna, July 1st, 2016.

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

[13] A. Wegener, M. Piorkowski, M. Raya, H. Hellbrück, S. Fischer, J.-P. Hubaux, "TraCI: An Interface for Coupling Road Traffic and Network Simulators", CNS 2008, April 2008.

Radio Irregularity Model in OMNeT++

Behruz Khalilov, Anna Förster, Asanga Udugama

Sustainable Communications Networks, University of Bremen, Germany
Email: {bk | afoerster | adu }@comnets.uni-bremen.de

***Abstract*—Radio irregularity is a non-negligible phenomenon that has an impact on protocol performances. For instance, irregularity in radio range leads to asymmetric links that cause the loss of packets in different directions. In order to investigate its effect, the Radio Irregularity Model (RIM) is proposed that takes into account the irregularity of a radio range and estimates path losses in an anisotropic environment. The purpose of this paper is to provide details of the RIM model developed in the INET Framework of the OMNeT++ simulator that can be used to investigate the impact of radio irregularity on protocol performance.**

I. Introduction

Anisotropic properties of the propagation media and the heterogeneous properties of devices are two major sources of radio irregularity that leads to unstable communication between two or more devices. The impact of radio irregularity on protocol performances can be investigated either through a running system or via a simulation environment. The first approach is not feasible due to its complexity. Therefore, simulation environment is widely used for this purpose. However, when considering radio propagation, most of the existing simulation models assume a spherical radio pattern that leads to an inaccurate estimation. The authors of [1] have proposed the Radio Irregularity Model (RIM) to cater for such irregularity that can be used for evaluation of an actual communication range of a sensor node dependent on medium and illustrate the variation of path loss in different directions. A key parameter of RIM is the Degree of Irregularity (DOI). DOI is used to describe the irregularity of the radio range and is defined as "*the maximum path loss percentage variation per unit degree change in the direction of radio propagation*" [1] (Figure 1). A further aspect of RIM is the use of the Weibull distribution [2] to model the variance of received signal strength of the different directions. The INET Framework of OMNeT++ provides a number of radio propagation models. The work presented here describes the RIM simulation model developed to supplement the other propagation models available in the INET Framework.

II. INET Implementation of RIM

The RIM functionality, i.e., **RIMFading**, is implemented in the INET Framework[1] by extending the **FreeSpacePathLoss** model. It contains the following functionality.

1) K_i, the coefficient of irregularity is generated once at the start of the simulation run for each node.

[1]INET Framework website: https://inet.omnetpp.org/

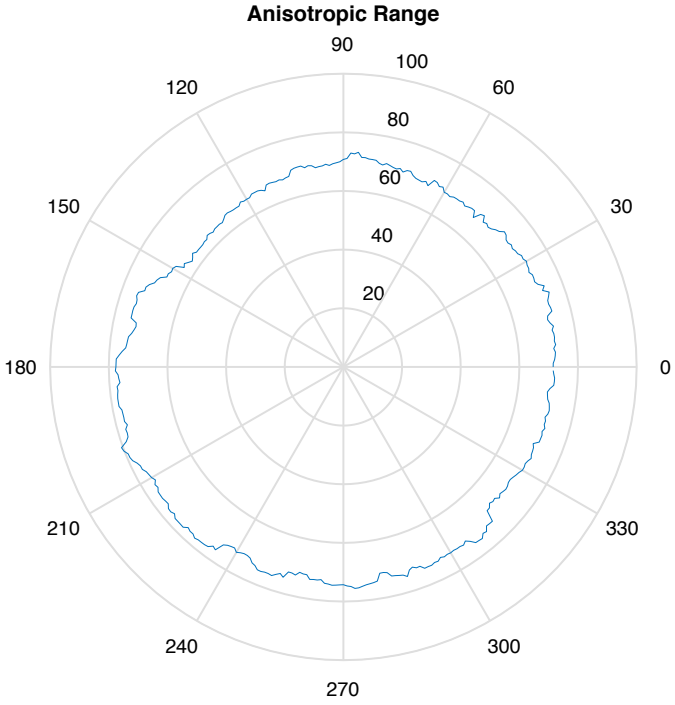


Fig. 1. Path loss variations per unit degree change in the direction of radio propagation

The iteration always starts from a constant point and this point is north or origin point of a transmitter:

$$K_i = \begin{cases} 1; & i = 0. \\ K_{i-1} \pm Rand * DOI; & 0 < i < 360^0 \end{cases}$$

$$|K_0 - K_{359}| \leq DOI$$

2) Compute path loss using Free Space Path Loss (FSPL). Any other path loss or fading models can be used instead of the FSPL model.
3) Calculate angle(s) between sender and receiver. By default RIMFading uses 2D model since the function that calculates the angle considers xy-plane and provides the value of the resulting angle for obtaining corresponding K_i value.
4) Multiply FSPL value with the corresponding K_i to obtain: $DOIAdjustedPathLoss = PathLoss * K_i$

Random numbers are generated using the Weibull distribution. Statistical analysis of empirical data collected from experiments [1] have shown that the variation of received signal strength in different direction fits the Weibull distribution.

Figure 2 shows the class inheritance diagram of the RIM implementation. It extends the functionality of the FSPL model to include the RIMFading.

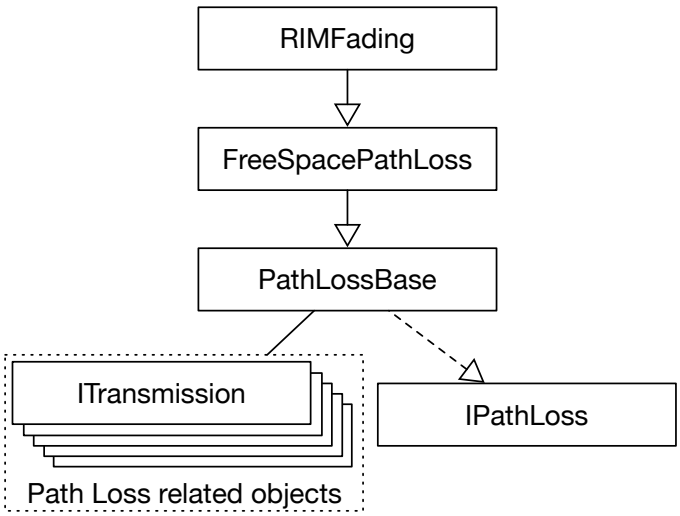


Fig. 2. Class inheritance diagram of the RIM implementation

III. Parameters

The RIMFading model has a number of configurable parameters that are defined in the NED file:

- **a**: The scale parameter of the Weibull distribution (default is 1.5).
- **b**: The shape parameter of the Weibull distribution (default is 1).
- **DOI**: The degree of irregularity (default is 0.006).

The default values for the shape (**a**) and scale (**b**) parameters of the Weibull distribution are selected to generate the required range of random numbers that are used to compute the coefficient of irregularity. The default value of **DOI** is based on the recommendation given in [1].

IV. Conclusion

The work presented in this paper relates to the development of the RIM simulation model for the INET Framework to simulate the variance of path loss with respect to the direction of propagations. The model developed in this work is released at Github (**https://github.com/ComNets-Bremen/RIMFading**) under a GPL License.

References

[1] G. Zhou et al, *Models and Solutions for Radio Irregularity in Wireless Sensor Networks*, ACM Transactions on Sensor Networks 2(2):221-262, May 2006

[2] W. Weibull, *A statistical distribution function of wide applicability*, Stockholm, Sweden, 1951

Discovering Neighbor Devices in Computer Network

Development of CDP and LLDP Simulation Modules for OMNeT++

Vladimír Veselý, Tomáš Rajca
Department of Information Systems, Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
veselyv@fit.vutbr.cz, xrajca00@stud.fit.vutbr.cz

***Abstract*—The purpose of data-link layer discovery protocols is to provide the network administrator with the current information (i.e., various Layer 2 and 3 parameters) about neighbor devices. These protocols are invaluable for network monitoring, maintenance, and troubleshooting. However, they start to play an important role in the operation of data-centers and other high-availability networks. This paper outlines design, implementation and deployment of Cisco Discovery Protocol and Link Layer Discovery Protocol simulation modules in OMNeT++ simulator.**

Keywords—data-link layer discovery protocols, CDP, LLDP, INET, ANSAINET

I. Introduction

Layer 2 discovery protocols have been developed to share information between directly connected devices. They send specific device's information (e.g., device role, interface state, assigned IP address, operating system version, Power over Ethernet capability, duplexness, VLAN configuration, etc.) to neighbors. These protocols are useful during network maintenance and process of troubleshooting when the administrator is trying to locate the source of a problem and isolate its layer presence. Cisco Discovery Protocol (CDP) was the first one from this family of data-link layer discovery protocols. CDP usage is limited to Cisco devices only due to its proprietary nature. Other vendors decided to follow the idea and developed their variants such as Foundry Discovery Protocol by Brocade, Bay Network Management Protocol and Nortel Discovery Protocol by Nortel, Extreme Discovery Protocol by Extreme Networks, Link Layer Topology Discovery by Microsoft, and others. In order to offer a multi-vendor environment, IEEE came with unifying protocol offering the same functionality as above mentioned representatives. It is codified in IEEE standard 802.1AB and called Link Layer Discovery Protocol (LLDP).

One of OMNeT++'s most popular frameworks is INET [1] that aims at providing models for Internet devices, protocols, and a mechanism to help with network design and configuration testing and evaluation. The Automated Network Simulation and Analysis for Internet Environment (ANSAINET) project is dedicated to the development of a variety simulation models compatible with RFC specifications or referential implementations, which extends the standard INET framework.

Both CDP and LLDP are de facto industry standards when it comes to network operation life-cycle. Since our goal is to develop simulation models for various networking technologies, we have decided to extend the functionality of ANSAINET. Hence, the paper outlines processes of adding support for CDP and LLDP, and it should be treated as finalized software contribution rather than research effort.

This paper has following structure. Section II covers a quick overview of existing implementations. Section III describes the operational theory and implementation design notes. Section IV contains testing scenarios. The paper is concluded in Section V, which also outlines our future work.

II. State of the Art

This section briefly overviews existing CDP and LLDP implementations for hardware/software routers and also simulators.

Since CDP is Cisco's intellectual property, CDP deployment in hardware is limited to Cisco's product portfolio only. Scarce CDP availability exists for simulators too. Cisco Packet Tracer [2] allows CDP configuration since its earliest versions. However, Cisco Packet Tracer is closed and proprietary simulator used mainly as an education tool.

On the other hand, LLDP is supported by a wide range of networking equipment vendors (e.g., Juniper, Hewlett-Packard, Arista, Brocade, including Cisco, and others) and operating systems (both Windows and Unix-based). We are not aware of any CDP/LLDP support by NS2/3 or OPNET.

During the ANSA project run, we have extended available simple network node with additional functionality – support for various routing, switching and data-link layer discovery protocols. The resulting `ANSARouter`, `ANSASwitch`, and `ANSAHost` components are a compound modules integrating all expected functionality in programmable simulation modules that adopt a Cisco-style representation of configuration, textual outputs (e.g., routing table format) and debugging information. This paper discusses CDP and LLDP implementation and their integration as new `networkLayer` submodules to `ANSARouter` and `ANSASwitch`. The simplified schema showing this integration in `ANSARouter` is depicted in Figure 2.

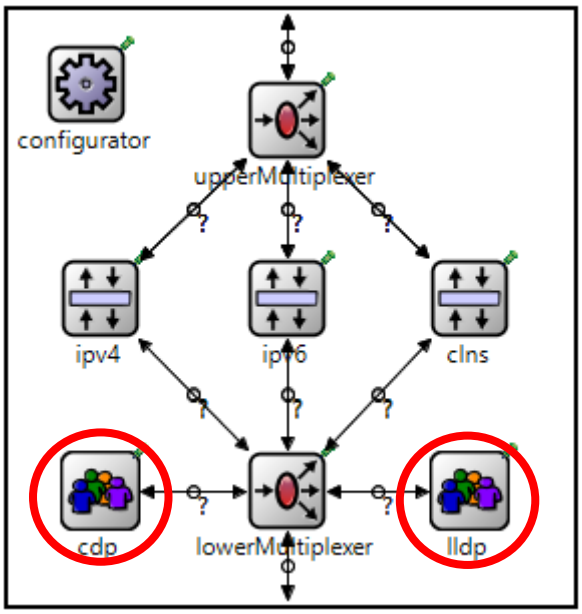


Figure 2: Structure of ANSARouter networkLayer

III. Principles

This section provides a description of principles of both CDP and LLDP. It includes the format of protocol messages and designed abstract data structures.

The reader is advised to follow references in order to learn more about particular protocol. CDP theory is based on references [3], [4], and [5]. LLDP theory is covered in sources [6], [7], [8], and [9].

A. Cisco Discovery Protocol

The current version 2 of CDP operates on any data-link layer technology with Subnetwork Access Protocol (SNAP) support, i.e., Ethernet, WiFi, Frame Relay, ATM or PPP.

CDP messages are sent to multicast MAC 01:00:0c:cc:cc:cc by default every 60 seconds. Data contained in CDP message are device dependent. CDP message consists of a generic header and a variable number of type-length-value (TLV) triplet fields. CDP header has following three mandatory TLVs – *Version*, *Time to Live*, *Checksum*. Shortened list of TLVs recognized by CDP is summarized in Table V, where columns marked "CDP TLV" and "TLV's Description" are relevant.

In OMNeT++, CDP is implemented as the compound module `CDP` interconnected with `lowerMultiplexer` of `networkLayer`. It consists of four submodules that are depicted in Figure 1 and briefly described in Table I. Our implementation is in full compliance with the observed behavior of Cisco's referential behavior.

B. Link Layer Discovery Protocol

LLDP operates in logical link control sublayer of data-link layer employing SNAP. LLDP terminology introduces agent, which is LLDP instance bound to a certain device's port. The agent sends and processes LLDP messages on a given interface.

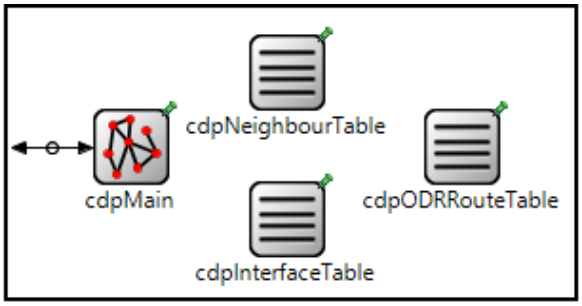


Figure 1: CDP module structure

TABLE I. DESCRIPTION OF CDP SUBMODULES

Name	Description
`cdp Main`	This module has core CDP functionality, which is responsible for parsing of XML configuration, message and timers handling, on-demand routing (ODR) functionality. Lightweight ODR is one of the main reasons we decided to implement CDP.
`cdp Neighbor Table`	This abstract data structure stores received CDP information from directly connected neighbors. Records are dynamically updated with every new CDP message received and expire after a given Time To Live value.
`cdp Interface Table`	Interface table contains a list of CDP enabled interfaces. This table state influences a periodic generation of CDP messages and included data.
`cdp ODRRoute Table`	This table holds routes learned via ODR extension. Each route is accompanied just like RIP with Invalid, Holddown and FlushedAfter timers.

LLDP data are stored in two management information bases (MIB) – first one local (for the device itself), second one remote (for information from neighbors).

LLDP message consists of a header with mandatory TLVs – *Chasis Id*, *Port Id*, *Time To Live* – followed by optional TLVs with additional data. All LLDP TLVs are included in Table V, where columns "LLDP TLV" and "TLV's Description" are relevant to LLDP. Additional TLV sets extending LLDP exist (e.g., LLDP-MED, DCBXP) but they are out of the scope of this paper. Comparing to CDP, LLDP optionally offers error management for its communication. Moreover, LLDP has also built-in rate-limiter for sending based on credit. LLDP standard assigns three dedicated multicast destination MAC addresses 01:80:c2:00:00:00, 01:80:c2:00:00:03, and 01:80:c2:00:00:0e (this one is default for Ethernet-based networks). LLDP message is periodically generated (by default) every 30 seconds.

We have similarly designed LLDP as CDP. `LLDP` compound module implements `INetworkLayerLower` interface, and it is interconnected with `lowerMultiplexer`. The module structure is depicted in Figure 3 and submodules description listed in Table II.

Figure 3: LLDP module structure

TABLE II. DESCRIPTION OF LLDP SUBMODULES

Name	Description
`lldp Main`	This module delivers core LLDP functionality. It sets up LLDP module based on XML preconfiguration. It governs sending and receiving of LLDP messages. It maintains neighborship and relevant information.
`lldp AgentTable`	Functionality is comparable with `cdpInterfaceTable` in sense that it contains interface specific LLDP settings.
`lldp NeighborTable`	The functionality of this abstract data structure is analogous to `cdpNeighborTable`.

IV. Validation and Verification

This section contains information about verification and validation of implemented simulation modules over the same set of scenarios. Demonstration example is purposely too basic, but both protocols have also been verified on more complex topologies.

Verification was conducted using a traditional approach employing code review, debugging and documentation [10]. We have found out that simulation models comply with their corresponding specifications; namely, the format of messages, configuration parameters meaning, and the functionality in all tested cases. In simulation validation, we have measured the accuracy of simulation models to real implementations on Cisco devices. As a part of this activity, we have set up same network scenarios in both simulator and the real environment. As a source of information, we analyzed packets exchanged between devices and debugging outputs of related processes. We built the test-bed environment from Cisco routers running IOS version 15.4(2)T4, Cisco switches running IOS version 15.2, and host stations with Windows 7.

Figure 4 shows the basic topology used for validation. It consists of three `ANSARouter` instances (marked R1, R2, and R3) and one `ANSASwitch` instance (marked S1) providing CDP/LLDP functionality and two `ANSAHost` instances (Host1 and Host2). To compare CDP and LLDP with each other, we have changed default LLDP timers – periodic generation of messages to 60 seconds and *Time To Live* value to 180 seconds.

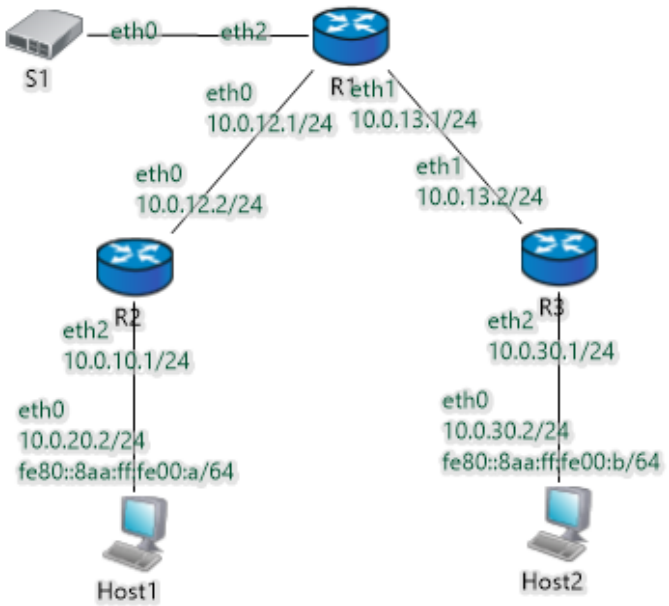


Figure 4: CDP/LLDP testing topology

Both protocol offer fast-start feature, which speeds up the process of neighbor discovery. During the fast-start, periodic message generation interval is just 1 second. Fast-start lasts for: a) three consecutive message updates in case of CDP; b) one to eight (by default three) consecutive message updates in case of LLDP. Fast-starts happens each time when: a) interface restarts in case of CDP; b) MIB content changes in case of LLDP standard; c) a new end-host is detected, or LLDP-MED TLV is exchanged in case of LLDP implementation by Cisco.

A. Initial Discovery

This test shows initial neighbor discovery when the interface changes from down state to up state after device successfully starts. We recorded all CDP/LLDP messages into PCAP file and compared timestamps. Table III shows the result for a link between R1 and R2 for both protocols (i.e., CDP and LLDP) and both simulated and real scenario.

Table III. Timestamp comparison for initial discovery

Direction	CDP		LLDP	
	Simul. [s]	Real [s]	Simul. [s]	Real [s]
R1 → R2	0.000	0.300	0.000	1.600
R2 → R1	0.000	5.370	0.000	1.900
R1 → R2	1.000	1.300	1.000	missing
R2 → R1	1.000	6.370	1.000	missing
R1 → R2	2.000	2.310	2.000	missing
R2 → R1	2.000	7.380	2.000	missing
R1 → R2	62.000	57.550	62.000	61.300
R2 → R1	62.000	66.850	62.000	61.400

B. Interface Restart

This test tracks events bound to the flapping of interface between R1 and R2. After the link goes down at $t = 50$s, records expire from tables at $t = 180$s. Then at $t = 200$s connection is reestablished and CDP/LLDP messages are first to appear on the wire. Table IV shows simulated and real scenario result for link between R1 and R2 for both CDP and LLDP.

Table IV. Timestamp comparison for interface restart

Direction	CDP		LLDP	
	Simul. [s]	Real [s]	Simul. [s]	Real [s]
R1 → R2	200.000	199.480	200.000	202.000
R2 → R1	200.000	201.500	200.000	205.000
R1 → R2	201.000	200.500	201.000	missing
R2 → R1	201.000	202.510	201.000	missing
R1 → R2	202.000	201.510	202.000	missing
R2 → R1	202.000	203.510	202.000	missing

C. Test Summary

We conducted multiple measurements on a real network, and the worst cases are depicted in Table III and Table IV, other runs were more accurate and aligned with starting event. The main causes of timestamp discrepancy are: 1) built-in jitters, which avoid alignment of several timeout events at the same time; 2) control-plane processing; 3) real device hardware processing. Different fast-start implementation by Cisco (which is not in compliance with the standard, see above) is the cause of missing LLDP messages in real network scenarios.

Validation discovered reasonable differences between our developed modules and referential implementation. The main goal of adding two new protocols to OMNeT++ was achieved.

V. Conclusion

In this paper, we briefly described two most deployed Layer 2 discovery protocols – CDP and LLDP. We created simulation modules of these protocols within OMNeT++ discrete-event simulator as new software contributions. We tested and verified functionality and accuracy of our models in comparison with the real network running referential implementation.

It is valuable to support CDP and LLDP within (ANSA)INET not only for the sake of completeness of simulated network behavior but also for any future research efforts. Both protocols already employ TLVs, which allow very convenient way how to add new functionality. Hence, our implementation offers a great starting point for any proof-of-concept extending original protocols. For instance, Software Defined Network related use-cases and technologies offer an interesting playground for our framework.

More information about the ANSAINET project is available on the homepage [11]. All source codes including CDP and LLDP implementations could be downloaded from GitHub repository [12].

Acknowledgment

This work was supported by the Brno University of Technology organization and by the research grant FIT-S-14-2299.

References

[1] OpenSim Ltd. (2017, June) INET Framework - INET Framework. [Online]. https://inet.omnetpp.org/

[2] Cisco Systems. (2017) Download Packet Tracer | Cisco NetAcad. [Online]. https://www.netacad.com/about-networking-academy/packet-tracer/

[3] Cisco Systems. (2015, May) Behavior of Cisco Discovery Protocol between Routers and Switches. [Online]. http://www.cisco.com/c/en/us/support/docs/network-management/discovery-protocol-cdp/118736-technote-cdp-00.html

[4] Cisco Systems. (2006) Catalyst Token Ring Switching Implementation Guide, Version 2. [Online]. https://docstore.mik.ua/univercd/cc/td/doc/product/lan/trsrb2/index.htm

[5] Cisco Systems. (2013, October) Chapter: Configuring Cisco Discovery Protocol. [Online]. http://www.cisco.com/c/en/us/td/docs/ios/12_2/configfun/configuration/guide/ffun_c/fcf015.html

[6] IEEE Std 802.1AB-2016. (2016) IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery. [Online]. http://standards.ieee.org/getieee802/download/802.1AB-2016.zip

[7] V. Z. Attar and P. Chandwadkar, "Network Discovery Protocol LLDP and LLDP-MED," *International Journal of Computer Applications*, vol. 1, no. 9, pp. 93-97, February 2010.

[8] Cisco Systems. (2006) LLDP-MED and Cisco Discovery Protocol. [Online]. http://www.cisco.com/en/US/technologies/tk652/tk701/technologies_white_paper0900aecd804cd46d.html

[9] Cisco Systems. (2014, August) Chapter: Configuring LLDP and LLDP-MED. [Online]. http://www.cisco.com/c/en/us/td/docs/switches/connectedgrid/cg-switch-sw-master/software/configuration/guide/sysmgmt/CGS_1000_Sysmgmt/sm_lldp.html

[10] Averill Law, *Simulation Modeling and Analysis*, Industrial Engineering and Management ed. New York, USA: McGraw-Hill, ISBN-10:0073401323, 2014.

[11] Brno University of Technology. (2017) ANSA Project. [Online]. https://ansa.omnetpp.org/

[12] GitHub/ANSA. (2017, December) kvetak/ANSA. [Online]. https://github.com/kvetak/ANSA

Table V. A list of CDP and LLDP TLVs

CDP TLV	TLV's Description	LLDP TLV
Version	CDP protocol revision number.	
	Unique identifier of the device in the scope of local area network, which may be derived from Layer 2/3 address, chassis or port component number, etc.	*Chassis Id*
Time To Live	Information is stored in a neighbor table for a period specified by this TLV record. For CDP, recommended value is 3× longer than a periodic generation; for LLDP, it is 4× longer.	*Time To Live*
Checksum	Message content integration check computed similarly as IP header checksum.	
Address	TLV contains sender's address. Optionally, it may carry also reflected recipient's address	*Management Address*
Capabilities	Specifies device's role within a network such as a router, switch, bridge, etc.	*System Capabilities*
Port-Id	String representation of sender's interface port label including index. This TLV is handy for checking the improper cabling	*Port Id*
	The label is specifying additional information about the interface for administrative purposes.	*Port Description*
Full/Half Duplex	Duplexness of sender's interface. This information may be used to detect duplex mismatch between devices	
Native VLAN	TLV hosts configured native (untagged) VLAN on a trunk interface. This TLV may be used to detect native VLAN misconfiguration	
Device-Id	Device's hostname (e.g., router1.local.lab)	*System Name*
Location	Device's topology location (e.g., Omega Bld., Rack 1)	*System Description*
Platform	Device's hardware descriptor (e.g., Catalyst 3560)	
Software Version	Device's operating system information usually as multi-line string representation	
VTP Management Domain	VLAN management extension governing the borders of another Cisco's proprietary protocol called VLAN Trunking Protocol	
IP Network Prefix	On-demand routing extension of CDP suitable for hub-and-spoke topologies. This TLV carries a list of device's network segments and configured default gateway	
	The last TLV in the list marking the end of LLDP message.	*EndOfLLDPDU*

OPS - An Opportunistic Networking Protocol Simulator for OMNeT++

Asanga Udugama, Anna Förster, Jens Dede, Vishnupriya Kuppusamy and Anas Bin Muslim

Sustainable Communication Networks, University of Bremen, Germany
Email: {adu | anna.foerster | jd | vp | anas1 }@comnets.uni-bremen.de

***Abstract*—The number of computing devices of the Internet of Things (IoT) is expected to grow by billions. New networking architectures are being considered to handle communications in the IoT. One of these architectures is Opportunistic Networking (OppNets). To evaluate the performance of OppNets, an OMNeT++ based modular simulator is built with models that handle the operations of the different protocol layers of an OppNets based node. The work presented here provides the details of this simulator, called the Opportunistic Protocol Simulator (OPS).**

I. Introduction

The Internet of Things (IoT) is expected to grow into a network of more than 50 billion devices by 2020 [1]. A communication architecture currently being considered for the IoT is Opportunistic Networks (OppNets) [2]. OppNets are a very versatile and effective means of communication to exchange data in a peer-to-peer manner. There are many application areas in the IoT that benefit from using OppNets. Areas such as social networking, emergencies are where information produced by user devices have a higher value locally than at other locations. OppsNets make it possible to propagate information locally to the interested parties through the different data dissemination protocols and the peer-to-peer link technologies used in most smart devices.

To evaluate the performance of OppNets, we have developed an OMNeT++ based simulator with a number of models. These models implement the functionality of the different layers of an OppNets based node. The purpose of this work is to describe the models available in this simulator, called the Opportunistic Protocol Simulator (OPS) to simulate OppNets.

The rest of this work is ordered in the following manner. The next section (Section II) provides a brief overview to OppNets that also includes an example use case. Section III provides a brief discussion on related work of OMNeT++ based OppNets implementations. Section IV provides the details of the models developed in OPS including the evaluation metrics available in OPS. Section V provides the details of some evaluations done using OPS. Section VI is a concluding summary with a look at future work.

II. Opportunistic Networks

The concept of Opportunistic Networking (OppNets) relates to peer-to-peer distribution of information in networks primarily without the help of any networking infrastructure [3]. Nodes which are deployed with protocols and mechanisms of OppNets communicate directly with each other when they come into contact.

OppNets are a very versatile and effective form of networking to be used in a number of application areas ranging from communications during disasters (e.g., South Asian Tsunami) to social networking. They operate on any peer-to-peer communication technology such as Bluetooth and IEEE 802.15.4. A key component of a node in OppNets is the forwarding (i.e., information dissemination) protocols used to propagate information in networks. There are a number of such protocols developed by researchers to efficiently disseminate information throughout a network (e.g., Epidemic Routing [4]).

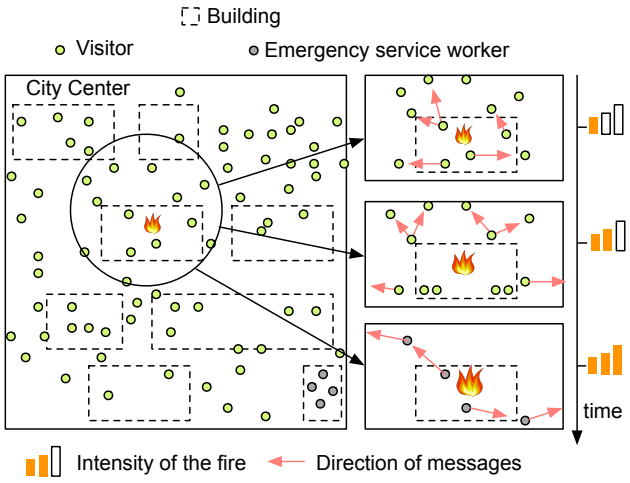


Fig. 1. An Example Use Case of OppNets: Dissemination of Information Related to an Emergency

Figure 1 shows an example use case of OppNets. A fire breaks out at a shop in the middle of the city center. The visitors and shoppers who are present at that location see this emergency and quickly moves away. The smart devices of these people propagates information related to this emergency over OppNets. This information reaches the emergency service workers who are then able to respond to the emergency.

III. Related Work

The OMNet++ simulator and its languages (NED and C++) provide the building blocks to build the layers of functionality of nodes in networks that operate using different wired and wireless protocols. Such protocol implementations are usually made available in OMNeT++ as frameworks. The INET framework is one such framework that provides the implementations for the protocols associated with the Internet Protocol (IP) suite. It is referenced by the official distribution of OMNeT++

and is recommended to be used when simulating IP based networks.

Though the INET framework consists of many protocols, it lacks the support for protocols required to simulate OppNets. A number of research efforts have focused on developing extensions to the INET framework and other publicly available frameworks to enable OppNets simulations [5]–[9]. Most of these works have concentrated on improvements to specific areas of OppNets (e.g., mobility, link layer, etc.) and therefore, has resulted in implementations that provide fine grained functionality in those specific areas, leaving out other areas. But, perhaps, the most important two drawbacks of these implementations are - firstly, these extensions have stayed unofficial and secondly, have become obsolete due to the architectural changes in OMNeT++ and the INET framework. Hence the necessity for developing a set of extensible models to build protocols for simulating OppNets in OMNeT++.

IV. Opportunistic Protocol Simulator (OPS)

The Opportunistic Protocol Simulator (OPS) is a set of simulation models in OMNeT++ to simulate opportunistic networks. It has a modular architecture where different protocols relevant to opportunistic networks can be developed and plugged in, to evaluate their performance.

The models of OPS are grouped into protocol layers of a protocol stack. The protocol stack represents the node architecture of a node in an opportunistic network. Each layer focuses on a specific area of operation of an opportunistic networking node. Figure 2 shows the node architecture of an OppNets node in terms of the protocol stack.

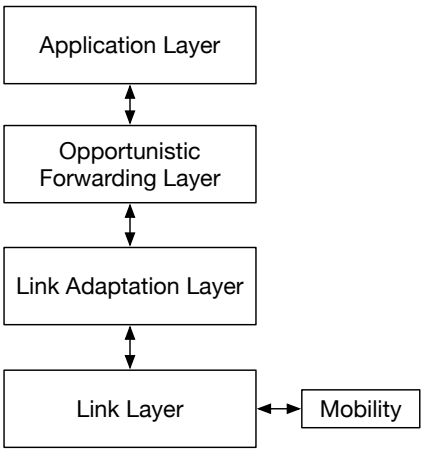


Fig. 2. Node Architecture of an OppNets Node

The **Application Layer** consist of models that generate data traffic in the network. Currently there are a few applications that were developed to evaluate the performance of OppNets with different data traffic generation patterns. These are listed below.

- **Promote** - This application focusses on generating random data that spread and publicise popular information in a network. A further evaluation done through this application was to understand how different data traffic generation models influence the performance of nodes in the network. There are 3 traffic generation models considered - constant traffic, uniformly distributed traffic and exponentially (poisson) distributed traffic.
- **Herald** - This application, though similar in its operation to the *Promote* application, generates a predetermined set of data in the network, instead of random data. Each data item is assigned a usefulness value by every node before the start of the simulation. The purpose of assigning the usefulness is to determine the percentage of useful data (or liked data) received by every node.
- **Bruit** - This application employs a simple traffic generation model where data traffic is generated using a uniform distribution.

In all these applications, the actual generation of data (i.e., injecting the data into the network) is assigned to a randomly selected node. A further capability of these applications is to generate destination oriented or destination-less data. In the latter case, the data is meant to simulate information that is liked by multiple users.

The **Opportunistic Forwarding Layer** is where any forwarding protocol related to disseminating data in opportunistic networks is plugged in. The generic operation of such a protocol is separated into the following parts.

- **Caching of Data** - Unlike in traditional networks, opportunistic networks are characterised by intermittent connections. Therefore, every node must employ a store-and-forward methodology when dealing with data.
- **Communicating with the Neighbourhood** - In opportunistic networks, the neighbourhood (i.e., nodes in the vicinity with which a given node can communicate) changes constantly. Therefore the forwarding protocol must employ mechanisms to pass-on data to maximize the propagation of this data in the network.

The current implementation of OPS includes 3 forwarding protocols. They are listed below.

- **Epidemic Routing** - Epidemic Routing [4] refers to a data propagation protocol where nodes negotiate with other nodes (in the neighbourhood) to determine unavailable data before the actual data is exchanged. Once a node knows what data it is missing in its cache, these data items are requested one after the other.
- **Organic Data Dissemination** - The Organic Data Dissemination (ODD) [2] refers to a technique of disseminating data based on the popularity of individual data items. It employs a mechanism to determine the significance of change of a node's neighbourhood (i.e., difference between the nodes arrived and left a neighbourhood with respect to a given node) to determine what kind of data is propagated. The *kind of data* refers to the popularity of the data - significant changes result in popular data being propagated.
- **Randomised Rumor Spreading** - Randomised Rumor Spreading (RRS) is a very simple forwarding protocol to spread data randomly in a network.

The *Epidemic Routing* protocol is inherently a *unicast* protocol while the other 2 are broadcast based protocols. The risk of *broadcast* protocols is flooding of networks and these 2 protocols have mechanisms that prevent the flooding of networks.

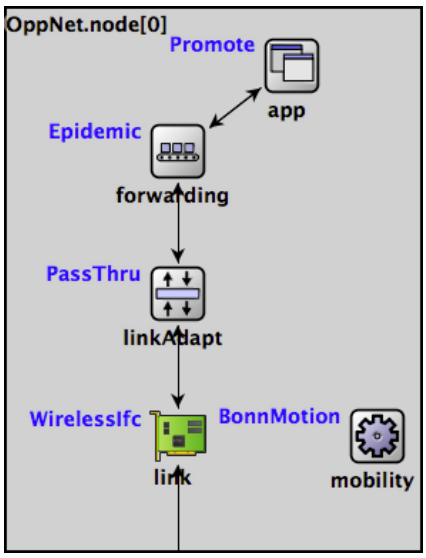


Fig. 3. An Example Node Configuration of an OPS Node in OMNet++

The **Link Adaptation Layer** is a layer where mechanisms that convert messages from one form to another is employed. An example is tunneling (de-tunnelling) of packes due to the end points of a communication spanning beyond the *Link Layer* end points. Currently, there are no specific adaptation mechanisms implemented. Hence, the **PassThru** implementation is used at this layer to let packets traverse this layer on its way between the *Opportunistic Forwarding Layer* and the *Link Layer*.

The **Link Layer** is where any link access technology is implemented. Currently, there is a modeled link layer implementation called **WirelessIfc**. This implementation, though not a full-fledged wireless link technology, models bandwidth, delays, wireless range and queuing present in a wireless link technology. It uses the Unit Disk Graph (UDG) method as the connectivity model. **Mobility** of nodes is handled using the mobility interface provided by the **INET Framework** of OMNeT++. *WirelessIfc* uses the position coordinates of nodes (retrieved from the mobility model) to determine the distance to other nodes. Since it uses the mobility interface of *INET*, it is able to use any mobility model implemented in *INET*. The *WirelessIfc* has been used to simulate *Bluetooth* network interfaces in our simulations [10].

The interfaces between each layer of the protocol stack has a standard format which has to be adhered to by all protocol implementations.

Figure 3 shows an example configuration of the protocol stack of a node in OPS. This configuration uses *Promote* application to generate data traffic. The data dissemination in the opportunistic network is performed using the *Epidemic Routing* protocol. *PassThru* is used to let packets through to other layers. *WirelessIfc* is used to perform wireless peer-to-peer communications and mobility is configured to use the *BonnMotion* mobility model of *INET*. The *BonnMotion* mobility model is a model to move a node using a trace file. A trace file provides the movement pattern in terms of X, Y, and Z coordinates.

OPS is programmed to provide performance data to compute a number of metrics. These metrics are very specific to OppNets to evaluate how well the data is disseminated in a network. The current metrics in OPS are listed below.

- **Liked Data Receipts** - This metric shows the amount of liked data received, compared to all the data received. Liked data are the data that were considered as being useful at the beginning of a simulation. The statistics are computed per node as well as for the whole network.
- **Non-liked Data Receipts** - This metric shows the amount of non-liked data received, compared to all the data received. Non-liked are the data that were not classified as being useful at the beginning of a simulation. The statistics are computed per node as well as for the whole network.
- **Data Traffic Spread** - This metric shows the Coefficient of Variation (CoV) which gives an indication of how well the data traffic is distributed in an opportunistic network.
- **Data Delivery Ratio** - This metric shows the ratio of delivered data to all the data that was generated.
- **Average Delivery Time** - This metric shows how long (at an average) it takes for data to be delivered to the intended recipient.
- **Average Contact Time** - This metric shows the average time that nodes were in contact during a simulation.
- **Number of Contacts** - This metric shows the times that nodes were in contact during a simulation.

As mentioned before, applications in OPS are able to operate as destination oriented or destination less data generators. Therefore, some of these metrics relate to only destination oriented data (e.g., *Data Delivery Ratio*).

V. Evaluations

OPS has been used in our research to simulate many OppNets based scenarios. [10] is a survey of simulators and concepts related to OppNets. This survey used OPS extensively and this section discusses some of the evaluations done with OPS.

Figure 4 is the result of simulations done to show the influence of different traffic generation models and cache sizes on the *Average Delivery Time* (referred to as Delay). The results are shown as a set of Cumulative Distribution Function (CDF) curves. The CDFs show that the traffic generation model employed has no influence on the delay. But, the cache size used to store the data influences the *Delay* significantly.

The simulations for this evaluation is done using a 50-node network. Each of these nodes uses the RRS forwarding protocol of OPS to disseminate data in the network. The

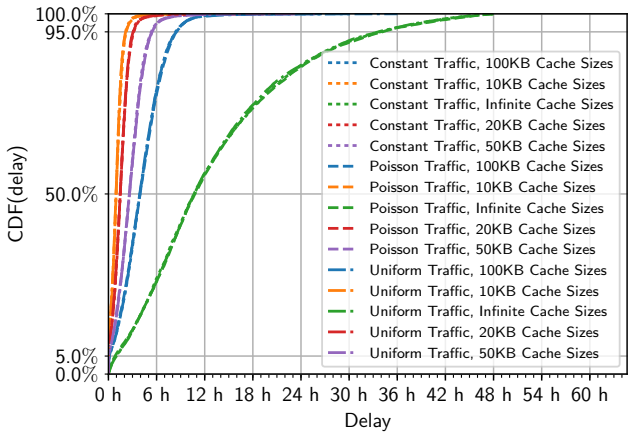


Fig. 4. CDF of the Delivery Delay of Data to Recipients using Different Traffic Generation Models and Cache Sizes

mobility of the nodes is performed using the *BonnMotion* mobility model of INET that is configured to use a 50-node extract of the San Francisco taxi cab trace (SFO trace) [11]. The simulation time was 30 days and the applications deployed in each of these nodes generate data with a mean interval of 2 hours.

Table I shows a comparison of the performance of the nodes in an OppNets network when using different mobility models, viz., a simple random model, a trace based model and a hybrid model implemented through Random Waypoint (RWP), Bonn Motion [12] and Small Worlds in Motion (SWIM) [13] [14], respectively. The Bonn Motion model uses the SFO trace. As before, the network consist of 50 nodes and a simulation duration of 30 days.

Model	RWP	SWIM	Bonn Motion
Simulation Time	4 min	59 min	109 min
Memory used	74 MB	86 MB	127 MB
Average Delivery Rate	3 %	96%	92 %
Average Delivery Delay	20.6 h	16.25 h	13.16 h
Total Number of Contacts	190	46,752	155,757
Average Contact Duration	117.14 sec	150.12 sec	584.39 sec

Table I. Performance results of different mobility models consisting of a simple random model (RWP), a trace based model (Bonn Motion) and a hybrid model (SWIM), with OPS

The applications deployed on the nodes generate data every 2 hours and the data carry a specific destination to which the data must be delivered. Further, the RWP and the SWIM models are configured as closely as possible to represent the behaviour of the SFO trace. The results show that mobility with real traces require a higer level of resources (simulation clock time and memory usage) compared to other mobility models. When considering the performance of the nodes, it is seen that the SWIM model provides the most closest results (Average Delivery Rate, Average Delivery Delay, etc.) to the trace based model. The trace based model is considered the most realistic due to the traces being collected from real mobility patterns.

The survey [10] referenced before included the performance results of other widely used OppNets simulators [15], [16] in addition to OPS. The results show that OPS provides a comparatively close performance to these simulators when considering the metrics listed in Section IV.

VI. Conclusion

The Opportunistic Protocol Simulator (OPS) is an OMNeT++ based modular simulator to evaluate the performance of OppNets. The models in OPS implement the functionality of the different layers of the protocol stack of an OppNets based node. The work presented here provides the details of the different models available in OPS.

OPS is used by us to simulate OppNets based scenarios [10] to evaluate the performance of dissemination protocols, traffic generation models, etc., to improve the performance of OppNets in the IoT. OPS is released at Github under a GPL License (*https://github.com/ComNets-Bremen/OPS*).

OPS is being extended with new functionality on a regular basis. Currently, there are a number of on-going projects focusing on areas related to forwarding protocols, applications, mobility models and user behaviour models. A long-term goal is to integrate OPS with the link technologies available with the INET framework.

References

[1] D. Evans, Cisco, *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*, April 2011

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

[3] L. Pelusi and A. Passarella and Marco Conti, *Opportunistic Networking:Data Forwarding in Disconnected Mobile Ad Hoc Networks*, IEEE Communications Magazine, November 2006

[4] A. Vahdat and D. Becker, *Epidemic Routing for Partially-Connected Ad Hoc Networks*, Technical Report, June 2000

[5] O. R. Helgason and K. V. Jónsson, *Opportunistic Networking in OMNeT++*, SIMUTools 2008, March 03 - 07, 2008, Marseille, France

[6] O. R. Helgason and S. T. Kouyoumdjieva, *Enabling Multiple Controllable Radios in OMNeT++ Nodes*, SIMUTools 2011, March 21 - 25, 2011, Barcelona, Spain

[7] S. T. Kouyoumdjieva et al, *Caching Strategies in Opportunistic Networks*, WoWMoM 2012, June 25 - 28, 2012, San Francisco, U.S.A.

[8] R. Zhang et al, *OppSim: A Simulation Framework for Opportunistic Networks Based on MiXiM*, CAMAD 2014, December 1 - 3, 2014, Athens, Greece

[9] Z. Zhao et al, *Performance Evaluation of Opportunistic Routing Protocols: A Framework-based Approach using OMNeT++*, LANC 2012, October 4 - 5, 2012, Medellén, Colombia

[10] J. Dede et al, *Simulating Opportunistic Networks: Survey and Future Directions*, IEEE Communications Surveys and Tutorials, Accepted for publication in 2017

[11] Michal Piorkowski at al, *CRAWDAD dataset epfl/mobility (v. 20090224)*, downloaded from http://crawdad.org/epfl/mobility/20090224, https://doi.org/10.15783/C7J010, February 2009

[12] N. Aschenbruck et al, *BonnMotion - A Mobility Scenario Generation and Analysis Tool*, SIMUTools 2010, March 15 19, 2010, Torremolinos, Malaga, Spain

[13] A. Mei and J. Stefa, *SWIM: A Simple Model to Generate Small Mobile Worlds*, IEEE INFOCOM 2009, April 19 - 25, 2009, Rio de Janeiro, Brazil

[14] A. Udugama et al, *Implementation of the SWIM Mobility Model in OMNeT++*, OMNeT Community Summit 2016, September 15 - 16, 2016, Brno, Czech Republic

[15] A. Keränen et al, *The ONE Simulator for DTN Protocol Evaluation*, SIMUTools 2009, March 2 - 6, 2009, Rome, Italy

[16] N. Papanikos et al, *Adyton: A network simulator for opportunistic networks*, [Online]. Available: https://github.com/npapanik/Adyton, 2015

Reactive User Behavior and Mobility Models

Anna Förster, Anas Bin Muslim, Asanga Udugama
University of Bremen, Germany
Sustainable Communication Networks Group
Email: [afoerster,adu]@comnets.uni-bremen.de

***Abstract*—In this paper we present a set of simulation models to more realistically mimic the behaviour of users reading messages. We propose a User Behaviour Model, where a simulated user reacts to a message by a flexible set of possible reactions (e.g. ignore, read, like, save, etc.) and a mobility-based reaction (visit a place, run away from danger, etc.). We describe our models and their implementation in OMNeT++. We strongly believe that these models will significantly contribute to the state of the art of simulating realistically opportunistic networks.**

I. Introduction

The research area of opportunistic and device-to-device communications has seen a very large growth in recent years. Applications and services are arising all over the world and begin to offer a real alternative to infrastructure-based ones. However, their testing and evaluation is extremely tedious and complex, as it relies on end users, on their behaviour and mobility patterns. Researchers have been using simulation for this purpose for a long time, but the existing simulation models are very restricted in mimicking real users.

Current user mobility models assume that people move in their environment without taking into consideration the data they receive through the network. Let us explore one of the mostly used motivational scenarios for opportunistic networks: disaster alert. When an accident or a natural disaster happens, like a fire or an earthquake, the application under test is supposed to send alert messages via device-to-device communications to all people around to warn them. In a real network, the reception of such a message will result in people running away. In a state-of-the-art simulation people will continue moving around as nothing has happened. This is the first problem we target with this work:

Goal 1: Users should react to the application messages in an appropriate way and change their moving pattern.

Furthermore, in current simulations, the user is assumed to produce and receive some messages without any meaning. In some cases the user provides preferences and wants to receive only a subset of all messages. However, the behaviour is also highly deterministic, which does not correspond at all to real human behaviour. This is our second goal:

Goal 2: Give meaning to the messages exchanged and provide the simulated user with an ability to react to these messages and to act non-deterministically.

In this paper, we propose a suit of simulation models to target the above problems. They are implemented as part of the new Opportunistic Protocol Simulator (OPS) in OMNeT++[1][1]. Section II describes our user behaviour model. Section III focuses the parameterisation of our model for various application scenarios. Section IV describes the corresponding reactive mobility model, while Section V discusses the metrics to be used with our model for evaluating opportunistic networks. Section VI describes the implementation of the models under OMNeT++, while Section VII finally concludes the paper.

II. Reactive User Behavior Model

Our model is based on the idea that the user reacts to all messages she sees in the system, e.g. she might ignore or delete the message, she might "like it", she might additionally comment on it or she might even save it for later reference. Which reaction the user has depends on the application, on the type of messages, on the number of messages and on the interests of the user. It also depends on the level of activity of this particular user - some users tend to comment on many messages, some react only sporadically.

Definition 1. *A* USER *is defined as a tuple* $u = (INT, R, base)$*, where* $INT = \{i_1, ...i_m\}$ *are the* m *interests of this user, stored as keywords,* $R = \{r_1, ...r_n\}$ *are the possible reactions of the user and* $base = Pr[X = r_i]$ *is the probability of this user to select a particular reaction* r_i*.*

With other words, the user is identified by her interests, her possible reactions to messages and her general attitude of reacting to messages. The $base$ probability should be a heavy tail distribution, e.g. a log-normal distribution. Such distributions have been shown many times to model well the behaviour of people, e.g. how many friends they have [2]. Examples are provided in Section III. The possible reactions are assumed to be an ordered set, where the first reaction is typically ignoring/deleting a message, while the next ones represent "heavier" reactions. For example, the user could vote (like) a message or save it for later. The last reaction is assumed to be the maximally possible reaction.

Definition 2. *A* MESSAGE *is defined as a tuple* $msg = (KEYS, pop, start, end, addr, radius)$*, where* $KEYS = \{k_1...k_l\}$ *are the keywords associated with this message,* $pop \in [0..100]$ *is the popularity of this message in the complete network (also defined as percentage of people who will react maximally at it),* $start$*,* end *and* $addr$ *are used for messages*

[1] https://github.com/ComNets-Bremen/OPS

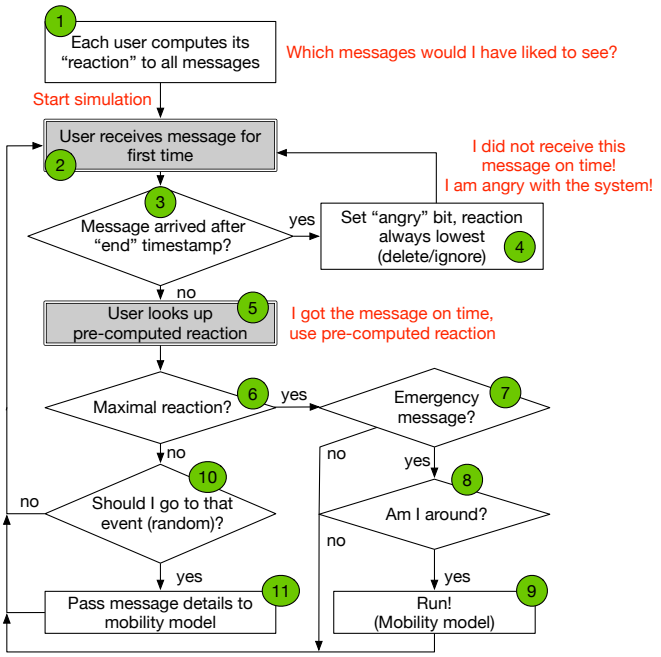


Fig. 1. General approach to compute and use user reactions.

carrying information about events. The radius is used to represent the danger area for emergency messages.

Note that all parameters of msg are optional. In the extreme case msg does not carry any meaning nor information, which does not make sense in the real world. Thus, either $KEYS$ or pop or $start, end, addr$ should be present. Only emergency messages have the $radius$ field.

The general flow of our model is depicted in Fig. 1. Note that the data dissemination protocol itself is not part of it. What kind of information it is using and how remains out of scope of this paper. In the next sub-sections, we will focus on the individual steps from Fig. 1.

A. Pre-Computing Reactions to Messages

The intuition behind our model is that there will be messages, which the user receives on time and takes some decision what to do and there will be messages, which arrive too late and are of no use any more. However, the user might have been interested in them (e.g., missed a concert of the favourite band). In real user experiments, we would ask the user *after* the experiment: Which messages would you have liked to see? For simulation efficiency purposes, we pre-compute these interests before we start the simulation and to look them up later.

The computation itself (step 1 in Fig. 1) depends on three parameters: the base activity level of the user, the popularity of a particular event and the matching keywords between the user and the individual event. While the exact weight of these parameters can be changed and fitted to real-world applications and users' behaviour, the general assumption is that a user would react more significantly to more popular messages (e.g. good jokes) and to events which better match her interests.

In our model, each reaction is associated with a selection probability: the base activity of the user. An example is

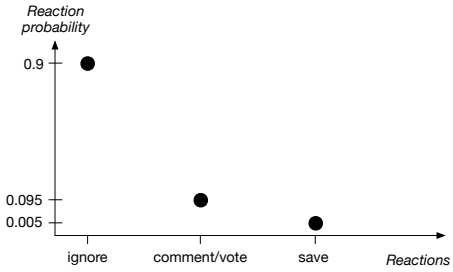


Fig. 2. Sample base activity probability distribution of a user. There are three possible reactions in total and most of the messages get ignored.

provided in Fig. 2. There are three possible reactions, IGNORE, VOTE/COMMENT, SAVE and the user ignores 90% of her messages, comments/votes on 9.5% of them and saves only 0.5%. This example is based on observations of real user data exchange of the Jodel application [2] over 3 days in Bremen, Germany.

If the popularity of the message is zero and there are no matching keywords with the interests of this user, the base probability is used directly to compute the reaction: in our example, the user will decide with 90% probability to ignore the message. If we represent these probabilities as differently spaced intervals and use a simple uniform random number generator, the intervals will look as in Fig. 3(a) and the random number is computed as:

$$r_{msg}^{user} = rand(0, 100) \tag{1}$$

In Fig. 3(a), the interval from which we can draw our random number is shaded and the probabilities of the individual reactions correspond exactly to the base probability of the user.

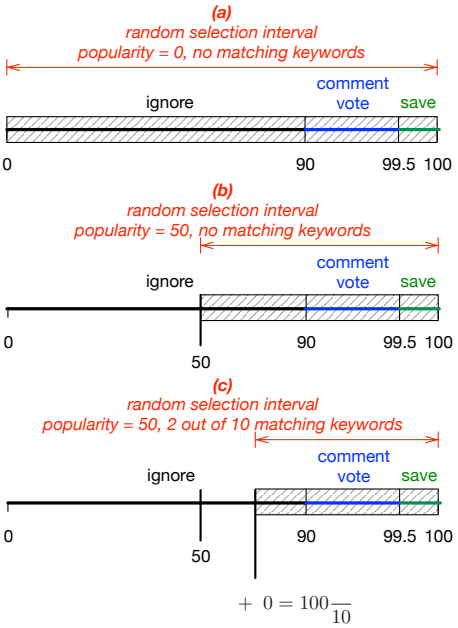


Fig. 3. Selecting the user's reaction for a message.

[2] https://jodel-app.com

If the message is more popular, we need to give preference to "stronger" reactions. We do this by limiting the interval, from which we can draw our random number, as shown in Fig. 3(b). This can be computed as:

$$r_{msg}^{user} = rand(pop_{msg}, 100) \quad (2)$$

where pop_{msg} is the popularity of this event. In this way, we limit the probability of selecting the lowest reaction and the probabilities of selecting one of the stronger reactions grow.

Furthermore, if there are matching keywords between the event and the interests of the user, we limit the possible selection interval further:

$$r_{msg}^{user} = rand(pop_{msg} + \frac{100k_{msg}^{user}}{l_{msg}}, 100) \quad (3)$$

where k_i^{user} is the number of matching keywords between this message and this user and l_{msg} is the total number of keywords of this message.

B. Mobility Reactions

The user could also react to a message with movement. For example, she could decide to go a concert or she might need to run away from a danger area. We consider two possible mobility reactions:

Immediate move: when the user receives a disaster alert and is in the area of danger (provided by the $radius$ field), she immediately moves out of the danger area. The exact location to which she moves is handled by the mobility model.

Scheduled move: upon reception of a message about a city event or similar, the user might decide to go there at the time of the event and passes this information to the mobility model.

As already noted in Section I, this mobility reactiveness of the user requires the mobility model to change immediately the movement pattern or to schedule a move to some place later on. We will discuss the implementation of this in Section IV.

C. Simulation

Once the reactions of all users to all messages have been precomputed, we are ready to start the simulation of our User Behaviour Model. It follows closely the flow chart of Fig. 1. For every message the user receives for the first time (step 2), the User Behaviour Model needs to react. It first checks whether this message has an end parameter and whether the message was received before this timestamp (step 3). If not, we mark this message with a special "angry" bit to be used later for computing the delivery rate of the system (step 4). Additionally, we always react to those late messages with the lowest reaction available (ignore/delete).

If the message arrived on time, we look up the pre-computed reaction (step 5). If that reaction was maximal (step 6), then we randomly decide whether the user will visit this event or not (step 10). If yes, $start, end, addr$ are passed to the mobility model to plan the move (step 11). A special case is provided when emergency messages arrive - if the user is in the danger zone of the emergency, she needs to exit the danger area immediately. Thus, the mobility model is told to "run away" (steps 8-9).

In case $start, end, addr$ are missing from the message, the flow becomes very short and only steps 1-2-5 are followed.

III. Application Model and Examples

In this section, we give some examples of applications and how to parametrise our user behaviour model accordingly. The proposed parameters are summarised in Table I. While this table suggests rather simple parameter sets, e.g. the same reaction probability for all users, it is of course possible to represent also more complex scenarios, e.g. different reaction probabilities for some more active users. Whether such complex scenarios deliver more realistic results is unclear and we will explore it in our future research.

a) Jodel-like Application: The first example we explored is a Jodel-like application, where users post anonymous messages to share with everyone in their vicinity. Usually those messages are jokes, general questions about life and studies, thoughts, etc. The presented parameter set is based on our observation of messages over 3 days in the area of Bremen. This set of reactions is used throughout this paper.

b) City Event Notifications: This example refers to a service, where users post messages about interesting events happening in a city, such as concerts, sports events, markets, etc. The users enter the time and place of the event, as well as some keywords. The proposed set of keywords is rather small to keep the scenario manageable. The set of users' reactions include an ignore option, a like option, a save option an a special option to save and go to the event.

c) Emergency Notification: This application targets the scenario, where a large-scale emergency happens, such as earthquake, large fire, tsunami, etc. The assumption is that everyone will read the message (for being informed or to be able to help) and, if in the danger area, will run away from it. The danger area itself is provided in the message.

IV. Mobility Model

Our mobility-enabled user behaviour model requires a mobility model, which is able to react to commands like "run away from X" and "schedule a move to Y at time Z". In principle, all existing mobility models can be changed to accommodate such commands. In the following, we describe the implementation of R-SWIM (Reactive SWIM), which is a reactive extension of our Small Worlds in Motion (SWIM) implementation [4].

R-SWIM, similar to the SWIM, keeps a track of neighboring loations and visiting locations (i.e., remote loations), and decides on the next destination using the SWIM equation. This procedure is overridden when the User Behaviour Model decides on a specific reaction (as specified in Section II). There are 2 types of reactions - *Immediate* and *Scheduled*. When an *Immediate* reaction is expected, R-SWIM identifies a new location to travel to based on standard SWIM and initiates the movement. When a *Scheduled* reaction is requested, R-SWIM queues the reaction to be executed when the appropriate time

TABLE I
SUMMARY OF SOME APPLICATIONS WITH THEIR PARAMETERS FOR THE USER BEHAVIOR MODEL.

Application	User			Messages				
	Number	Interests	Reactions & Probability	Number	Traffic (creation)	Keywords	Popularity	Time and place
Jodel-like campus	500-1000	none	Ignore (90%), comment/vote (9.5%), save (0.5%)	5 (day/user)	Poisson	none	0 (70%), 10-20 (29%), 50 (1%)	none
City event announcements	2000-10000	2-5 out of: sale, concert, exhibition, outdoor, food, happy hour, market, sports, demonstration	Ignore (80%), like (15%), save (4.5%), save&go (0.5%)	0.1 (day/user)	Poisson	2-5 out of: sale, concert, exhibition, outdoor, food, happy hour, market, sports, demonstration	0 (70%), 1-5 (29%), 10 (1%)	Place: mostly city center, spread some around. Time: mostly evenings and weekends.
Emergency notification	2000-10000	none	Read&run (if close) (100%)	0.1 (day/user)	Poisson	none	100 (100%)	Random

is reached. The appropriate time is decided by the mobility model based on the timing of the associated event and the distance to travel.

V. Evaluation Metrics

The real difference between traditional metrics and ours is what we understand under "delivered messages". Only messages, to which the user has reacted with more than "ignore" and were received on time are considered successfully delivered. All other messages, duplicates, etc. should be considered overhead. Otherwise we compute **delivery rate** and **delivery delay** as usual, taking into account the injection time of each message and its first delivery to the user. Furthermore, we compute **per user overhead** as a percentage between the number of messages this particular user has received over the number of her successfully received messages. All of the above mentioned metrics should not only be presented as means and confidence intervals, but also evaluated in terms of their **fairness distribution**. This is very useful to explore whether some users have been neglected in terms of data delivery and delay or overloaded with traffic.

VI. OMNeT++ Implementation

We have implemented the above described models in OMNeT++. The structure of the implementation is depicted in Figure 4. It is part of our Opportunistic Protocol Simulator (OPS) for OMNeT++.

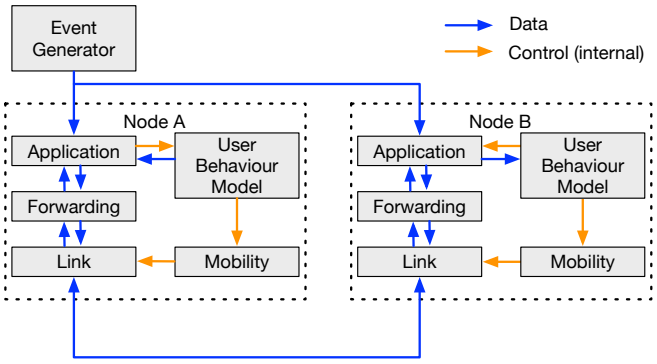


Fig. 4. Interactions between the individual modules in OPS.

The behaviour of the models UserBehavior, Mobility follow the here presented models. EventGenerator is the module, which injects the messages into nodes. Application is rather a place-holder for more complex application scenarios and in this case simply forwards new messages to the UserBehavior. The other models are part of the OPS Framework.

VII. Conclusion and Future Works

In this paper we have mostly used our experiences with real users, social traces and expectations. The presented models show very promising results and are much more valuable for researchers than existing simulation models. However, we have not confirmed the user simulation parameters (e.g. the probability function to rate events) with real user traces. We are planning to do this in the immediate future.

Furthermore, we plan to extend the user behavior models with other applications and environments, e.g. weather forecast, transportation, etc. We are happy to cooperate with other research groups on these topics.

References

[1] A. Udugama, A. Förster, J. Dede, V. Kuppusamy and A. Bin Muslim, *OPS - An Opportunistic Networking Protocol Simulator for OMNeT++*, Proceedings of the OMNeT++ Summit, 2017.
[2] M. Newman, *Networks: An Introduction*, Oxford University Press, 2010.
[3] A. Förster, A. Udugama, C. Görg, K. Kuladinithi, A. Timm-Giel and A. Cama-Pinto, *A Novel Data Dissemination Model for Organic Data Flows*, 7th EAI International Conference on Mobile Networks and Management (MONAMI), Santander, Spain, 2015.
[4] A. Udugama, B. Khalilov, A. Bin Muslim and A. Förster, *Implementation of the SWIM Mobility Model in OMNeT++*, Proceedings of the OMNeT++ Summit, 2016.

Parameterization of the SWIM Mobility Model Using Contact Traces

Zeynep Vatandas, Manikandan Venkateswaran, Koojana Kuladinithi, Andreas Timm-Giel
Institute of Communication Networks, Hamburg University of Technology, Germany
Email: {zeynep.vatandas | manikandan.venkateswaran | koojana.kuladinithi | andreas.timm-giel}@tuhh.de

***Abstract*—Opportunistic networks (OppNets) are focused to exploit direct, localised communications which occur in a peer-to-peer manner mostly based on people's movements and their contact durations. Therefore the use of realistic mobility models is critical to evaluate the data dissemination in OppNets. One of the mobility models that is available in OMNeT++ which can be used to mimic human movement patterns is Small Worlds in Motion (SWIM). The SWIM model is based on the intuition that humans often visit nearby locations and if the visited location is far away, then it is probably due to the popularity of the location.**

As an alternative to mobility of a node, pairwise contact probabilities are also used to evaluate the data dissemination in OppNets. Pairwise contact probabilities can be used to predict that a node will be met by a particular node. These probabilities can be derived in many ways. One of the ways is to calculate the average probability with which a node will meet another particular node at any point of time. Another way is to calculate the probability with which a node will meet another based on the time of day. The way of calculating pairwise contact probability depends on the scenario.

In this work, the pairwise contact probabilities obtained from the real traces are used to tune the parameters of the SWIM mobility model. The traces and the SWIM model are compared in terms of contact durations, inter-contact times and, number of pairwise contacts. How to decide SWIM parameters using real contact traces are being addressed.

I. Introduction

Opportunistic networks use the mobility of users (or nodes) to send data by storing and forwarding techniques. The data transfer is done when there is a contact between two users. In other words, data is transmitted only when two nodes are within their communication range. The communications are often between short range nodes and, ad-hoc in nature. Since the network is delay tolerant, the data transfers are often used for non-critical situations. When evaluating the performance of OppNets, OMNeT++ plays a major role as a network simulator due to the scalability and availability of different OppNets routing and mobility models. [5]

This paper mainly deals with the simulation of opportunistic networks using human mobility models. To simulate human mobility, lots of different mobility models are proposed till now. Since pure random models are not good and realistic, traces are difficult to obtain, the authors of [2] have proposed a simple self tuning mathematical model to model human behaviour, called as Small Worlds in Motion (SWIM) which is based on location preferences alone. In this paper, data from real life traces taken by students from the Cambridge University [4] are analysed and inferences from the traces are used to decide on the parameters for simulating the SWIM model. The Cambridge traces are Bluetooth traces recorded at various scenarios and locations. Moreover, methods to calculate pairwise contact probabilities are analysed and the influence of the pairwise contact probabilities on choosing the simulation parameters are discussed. The pairwise contact probabilities are represented as a matrix and ways to calculate the matrix are discussed. Based on the parameters derived from traces, the SWIM model is used in our scenarios to compare different performance metrics (such as Inter-contact times, contact durations, number of pairwise contacts, number of contacts per unit time and, pairwise contact probabilities). By comparing these features related to mobility patterns of humans, properties present (and not present) in the SWIM model are discussed. Therefore, the work presented in this paper provides how SWIM parameters can be adjusted to mimic the same characterestics of contact based traces in a given scenario. We use the SWIM mobility model [6] available in OMNeT++ to validate SWIM parameters referring three different traces.

The rest of this paper is ordered in the following manner. The next section (Section II) provides a brief introduction to the SWIM mobility model and the contact traces. Section III provides how to select SWIM parameters and description of the scenario. Section IV is a validation of the results taken using the SWIM mobility model and the pure contact traces. Section V is a concluding summary.

II. Overview: SWIM Mobility Model and Contact Traces

A. SWIM Mobility Model

SWIM [2] 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.

Each node in SWIM has a home location permanently assigned to itself. The home location is chosen randomly from the simulation network area. The nodes can move only to certain number of locations which are scattered around the network area randomly. Other than these locations, a node can move to its own home and each location is treated as a square

cell C. Each node maintains a weight w(C) value for locations in the map. A node does not need to have a weight for each and every location in the map. A node only maintains weights for the locations it has visited. At the beginning, the weights will be initialised to 0. The weight of a location can be calculated as shown in equation (1) below.

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

The equation represents the weight that node A assigns to cell C. In this equation, $distance(h_A, C)$ is a measure which decays based on power-law of distance from node A's home to cell C, α is a constant value in the range of between 0 and 1 and $seen(C)$ is the number of encountered nodes at cell C by node A and $seen(C)$ is updated 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.

According to [6], SWIM implementation in OMNeT++ is done by extending the *LineSegmentsMobilityBase* class.

B. Contact Traces

The SWIM model was simulated by the authors in [2] and compared with Cambridge Bluetooth traces [4]. SWIM model has been implemented in ONE simulator and derivation of SWIM parameters was not clear. Therefore, we try to verify how to derive α value based on node pairs meeting probability in traces and to see how others parameters like neighborhood area should also be adjusted in OMNET++ to get the exact behaviour of mobility patterns using SWIM. The Cambridge traces [4] consist of different experiments done at different locations. We have used only 3 kinds of traces. They are;

- INFOCOM 2005: This was an experiment conducted at a conference at Miami inside a hotel in March 2005. This experiment had 41 mobile nodes carried by the attendees of the conference for 4 days.
- Cambridge 2005: This was an experiment conducted during October 2005 for 11 days in and around the Cambridge University by distributing iMotes to 36 students and 18 stationary iMotes placed at various locations such as labs, shops, pubs, etc.
- INFOCOM 2006: This experiment was conducted at a conference (April 2006) at Barcelona in a hotel with 78 attendees as mobile nodes and 20 stationary nodes. The experiment lasted for 5 days.

III. Scenario

Nodes do not attract each other in pure location based models. Locations attract the nodes. This creates a challenge in parameterization of the SWIM model using pairwise contact probabilities. If the nodes attract each other, the pairwise contact probabilities can be used directly to program the attraction between node pairs. A way must be found to translate the location preferences of a pure location based model into the pairwise contact probabilities derived from the real traces. In this section, we discuss an approach to derive the α for the SWIM model from pairwise contact probabilities.

The pairwise contact probabilities are calculated for the whole of the experiment time. The calculation starts with the number of contacts matrix which is a NxN matrix, where N is the number of nodes in the experiment. Let us call this matrix as A. The element in the i^{th} row and j^{th} column of the matrix A represents the number of pairwise contacts between the i^{th} and the j^{th} node throughout the experiment. This matrix has the leading diagonal elements as 0 as a node cannot contact itself. The matrix A is symmetric. The algorithm for calculating the pairwise contact probability matrix P from the matrix A is given below in equation (2). Pairwise contact probability can be calculated by normalising each element of the matrix A by the sum of the upper triangle of matrix A.

for each row i in matrix A

for each column j in the row i of matrix A

$$P_{ij} = \frac{A_{ij}}{\frac{\sum_{k=1}^{N}\sum_{l=1}^{N} A_{kl}}{2}} \quad (2)$$

Deciding the parameter (e.g. α in SWIM model) for a pure location based model from pairwise contact probabilities is not very direct as there is no way to exactly match the values of pairwise contacts between node pairs. Instead, an overall pattern is observed by sorting the upper triangle of the pairwise contact probability matrix P in ascending order. Figure 1 and figure 2 show the resultant matrix P_{pair} against the number of possible node pairs obtained from Cambridge and INFOCOM 2005 traces. These figures show two regions, a nearly linear increasing region at the start of the graph which is followed by a sudden increase region which breaks the linearity.

Higher the probability of visiting only nearby locations, higher will be the probability of preferring only a few nodes. A high α (say 0.9) will make sure that a node visits nearby locations 90 % of the time and the other locations outside the neighborhood are visited only 10% of the time. A low α allows a node to visit outside the neighborhood and hence increasing the probability of meeting all nodes. Based on the above intuitions, the following thumb rule is used to decide on the α.

The more linear the plot increases, lower is the α. A fully linear increasing plot nearly parallel to the X axis needs an α of zero to make sure that there is a chance to meet every node. The greater the slope of the non-linear sudden increase compared to the linear region, greater the α. The above thumb rule is under the assumption that the neighborhood radius with respect to home location of a node is small when compared to the whole map. If neighborhood radius is as big as (or comparable) with the map size, then a high α will not have any meaning since all locations in the map will be treated as nearby locations and the model will start behaving like a random waypoint model for α=1. In the Cambridge 2005 trace (figure 1), nearly half the node pairs did not meet each other (P_{pair} = 0) indicating the need for the nodes to stay within the neighborhood and avoid meeting all the nodes. The sudden rise in the P_{pair} (figure 1) denotes high probability of meeting for a few node pairs. The Cambridge 2005 trace needs a high

α and hence an α of 0.9 is used for simulating the SWIM model.

Greater the α, higher the graph rises for higher node pair numbers. For low α, the graph will be nearly flat without any non-linear region.

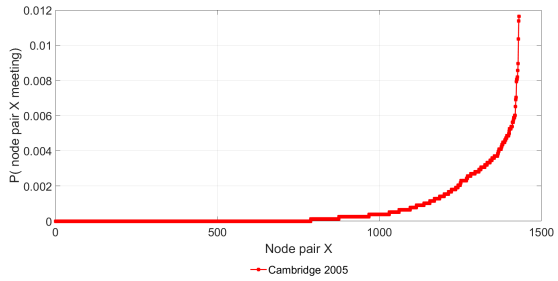


Fig. 1. CAMBRIDGE 2005 - Node pair meeting probability

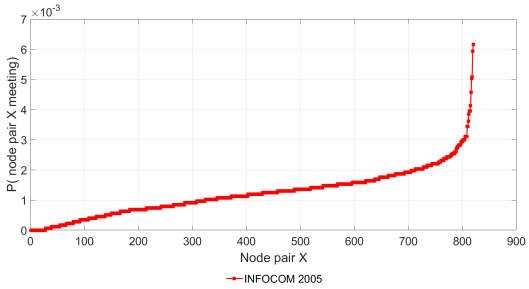


Fig. 2. INFOCOM 2005 - Node pair meeting probability

The detailed comparision of results are shown here with regard to only the Cambridge traces. The proposed α of 0.9 is the nearest match for the Cambridge 2005 trace. An α of 1 might also be the best match as it would reach even higher than α = 0.9. But the task is just to find out the way in which the pairwise node meeting probability behaves for different α values.

Table I lists the parameters used to configure a scenario with SWIM mobility model based on Cambridge trace.

Parameter	Cambridge Traces
Number of nodes	36 mobile nodes, 18 stationary nodes
initialX, initialY	Randomly selected values
maxAreaX, maxAreaY	2000 meters x 2000 meters
waitTime	exponential (30 minutes)
alpha (α)	0.9
noOfLocations	38 (Stationry nodes are placed in the locations randomly.)
Beacon Interval	Mobile nodes (10 min), 4 long range nodes (2 min), 2 short range nodes (6 min), 12 short range nodes (10 min)
Radio range	Mobile nodes (11m), Stationary - short (11m), long (22m)
Simulation time	11 days
Neighborhood radius	100m

TABLE I. USED PARAMETER VALUES

In this work, we analyse the node pair meeting probability which gives us a measure of how often node pairs meet. The node pairs do not indicate any particular node pair. Instead, the node pair numbers in the X axis just indicate the number of node pairs in the simulation as shown in figure 1, 2 and 3. The node pair meeting probability used in this work is useful in calculating the data dissemination time similar as in [7]. Assuming that we randomly assign exact node pair identities to the X-axis, the data dissemination time will not only depend upon contact duration and inter contact time, but, it also depends on the node which is met next. This is due to the simple fact that each node might have a different set of data at any given point of time and due to this, the node which will be met next will play a major role in data dissemination time. For example, in a situation where a Node-X meet only one other Node-Y during the whole experiment, it is totally possible that the Node-X will never have all the data in the network and the network may never converge towards a finite data dissemination time due to this Node-X.

It is important to note that, since we only try to analyse the effect of α values on pairwise contact probabilities, the other parameters are fixed such as waiting time parameters. The parameters other than α which are set as constant cannot represent the human mobility in the traces very accurately because they are derived from the aggregate data of all nodes in the mobility traces. Therefore, in this work, we only try to tune the most important deciding parameter of the SWIM model (α) using the pairwise contact probability.

IV. ANALYSIS OF RESULTS

The Cambridge 2005 trace is simulated with multiple α values and the parameters mentioned in Table I. Figure 3 shows that the proposed α of 0.9 matches the node pair meeting probability of Cambridge 2005 trace. The greater the α, higher the graph rises for higher node pair numbers. For low α values, the graph is nearly flat without any non-linear region. Figure 4 shows that the contact durations of Cambridge 2005 and SWIM model are almost a perfect match. Two nodes will be in contact if they are at rest at the same place within communication range or if they meet while in motion.

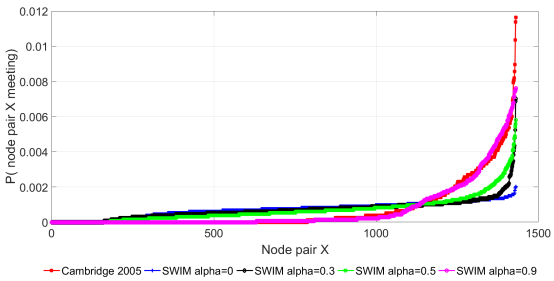


Fig. 3. Node pair meeting probability - Cambridge vs SWIM

The latter is not easily controllable as it depends on a lot of random movements. The wait times are the easiest way to tune the contact durations. Choosing the right distribution for the wait times plays a key role in achieving a good match for the contact durations. In SWIM model, since the nodes move with speed equal to the distance between the source and destination, the motion of the nodes contribute negligibly to

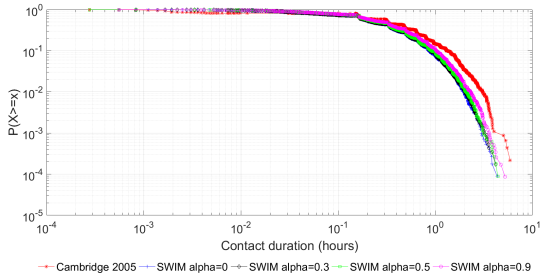


Fig. 4. Contact duration - Cambridge vs SWIM (log-log axis)

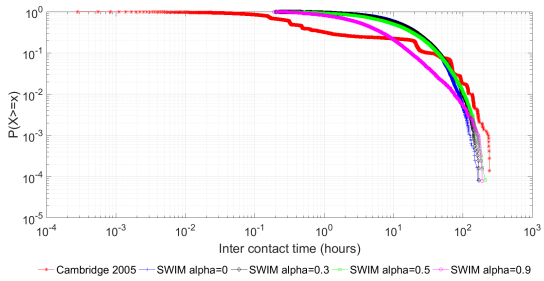


Fig. 5. Inter-contact times - Cambridge vs SWIM (log-log axis)

the contact durations. The wait times are the main contributors to contact durations.

In figure 5, the inter-contact times of SWIM simulation are very similar to the Cambridge 2005 trace. The graph follows a power law up to approximately 12 hours and followed by an exponential cut-off. In figure 6, we can see the day and night patterns of Cambridge 2005 and the absence of it in SWIM model.

Figure 7 shows the frequency of a node having zero number of contacts per hour is very high for Cambridge 2005 when compared to the SWIM simulations. To have zero contacts per hour per node, a node must not meet any node for one hour. This is highly probable only if a node is at rest. Movement increases the possibility of meeting another node. Since there is no assurance of rest in a periodic manner, the possibility of SWIM model having zero contacts per hour per node is not as high as Cambridge 2005. There is a lack of high number of contacts per hour per node in SWIM. In Cambridge 2005 trace in figure 1, some of the nodes have achieved more than 20 contacts per hour even though the frequency is very low. But, SWIM simulations did not achieve more than 10 contacts per hour per node for any α value. In real life traces, nodes have different wait time parameters and different α values. If the simulation is done with just one common α and wait time distribution for all the nodes, then it will not be possible to achieve low and high contacts per hour per node. To achieve zero contacts per hour per node, there must be a possibility to have longer wait times which enable the nodes to be less active. The lack of a night time removes the possibility of the nodes resting in a periodic manner which in turn reduces the possibility of having longer wait times. To achieve more contacts per hour, a node must either move quickly without waiting long at any location to meet other nodes within short durations or a node must be visited by a lots of other nodes within a short amount of time. This is possible only if the nodes are programmed in a heterogeneous manner in terms of wait time and α.

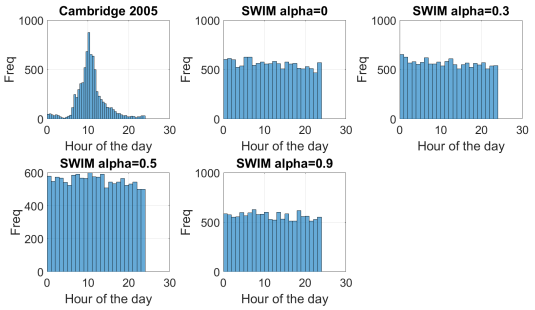


Fig. 6. Number of overall pairwise contacts based on hour of the day

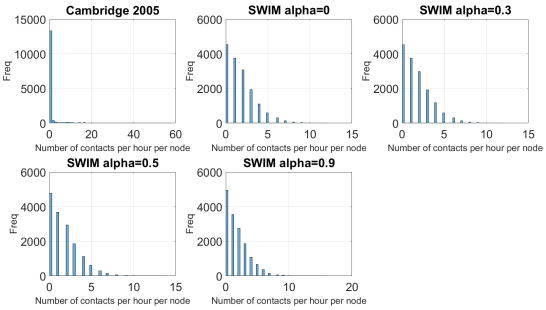


Fig. 7. Aggregate number of contacts per hour per node

V. Conclusion

In this work, pairwise contact probabilities are used to guess the alpha value. However, we do not yet have a mathematical method to derive an alpha value using the pairwise contact probabilities, we try to analyse the pattern in node pair meeting probability for different alpha values and propose a thumb rule for guessing an alpha value. The work in [2] introduces the SWIM model but the reason for alpha value to be chosen is not very clear. The unique contribution of this work comparing to the work in [2] is:

- We find a way to parameterize a location based model (SWIM) using pairwise contact probabilities extracted from real traces. In a location based model, we do not control the attraction between node. On the other hand, we have control over where a node will go using location preferences. In this work, we try to translate location preferences (neighborhood area and α) into node preference by sorting the pairwise contact probabilities. In this sorting process, node pair identities are lost and we use the remaining pattern to parameterize the SWIM model.

This work investigated how to tune SWIM parameters to replicate the pairwise contact probabilities of a very heterogeneous real life traces of Cambridge traces. The α values and the size of the neighborhood area have an effect on the mobility

of nodes. A larger α value results in selecting target locations close to the home location which in turn limits the number of nodes encountered. On the other hand, a smaller α value results in selecting popular locations which in turn increases the probability to meet all the nodes in the experiment.

Pairwise contact probabilities are parameters of both social and location attractions. Some people move to meet other people and some people move to go to places. The use a purely location based model like SWIM to match the pairwise contact probabilities of real life movements is a hard challenge. By design, SWIM model still lacks heterogeneity in terms of activity level, waiting times and other parameters such as neighbourhood radius and popularity decision threshold. Therefore, some of the further work include:

- Different α values: can be used to represent day and night time mobility behaviour. Nodes can be divided into clusters representing a particular behaviour with regard to α (e.g., behaviour of students vs teachers)
- Different neighbourhood radius: multiple sectors with different radii making each sector having a different priority of visiting. This allows the possibility of expanding the neighbourhood into fine divisions (e.g. Kitchen and lab area in Cambridge traces)
- A mathematical model: This work is starting point for using pairwise contact probabilities for parameterizing the SWIM model. Therefore, a mathematical model to predict the α and other parameters of SWIM model based on existing properties of real life traces (e.g. pairwise contact probability) is a part of future work. The approach presented in this paper is purely graphical in guessing the α for simulating the SWIM model.

REFERENCES

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

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

[3] 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

[4] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau, *CRAWDAD data set cambridge/haggle (v. 2006-01-31)*, http://crawdad.org/cambridge/haggle/20090529/

[5] *https://github.com/ComNets-Bremen/OPS*

[6] A.Udugama, B.Khalilov, A.b.Muslim, A.Foerstrer, K.Kuladinithi, *Implementation of the SWIM Mobility Model in OMNET++* Proc. of the 3rd OMNeT++ Community Summit, Brno University of Technology - Czech Republic - September 15-16, 2016

[7] K.Garg, S. Giordano, M. Jazayeri, *How Well Does Your Encounter-Based Application Disseminate Information?*, Conference: The 14th IFIP Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net) - 2015

A Remote Interface for Live Interaction with OMNeT++ Simulations

Maximilian Köstler* and Florian Kauer†
Institute of Telematics, Hamburg University of Technology
Am Schwarzenberg-Campus 3, 21073 Hamburg, Germany
*maximilian.koestler@tuhh.de, †florian.kauer@tuhh.de

***Abstract*—Discrete event simulators, such as OMNeT++, provide fast and convenient methods for the assessment of algorithms and protocols, especially in the context of wired and wireless networks. Usually, simulation parameters such as topology and traffic patterns are predefined to observe the behaviour reproducibly. However, for learning about the dynamic behaviour of a system, a live interaction that allows changing parameters on the fly is very helpful. This is especially interesting for providing interactive demonstrations at conferences and fairs.**

In this paper, we present a remote interface to OMNeT++ simulations that can be used to control the simulations while visualising real-time data merged from multiple OMNeT++ instances. We explain the software architecture behind our framework and how it can be used to build demonstrations on the foundation of OMNeT++.

I. Introduction

OMNeT++ [1] offers a variety of powerful visualisation tools that can also be used to display information at simulation time. However, many features that would be useful for live demonstrations with OMNeT++ are not available. This includes a convenient and clean interface to manipulate simulation parameters on the fly as well the possibility to merge multiple data sets from different simulations. The Institute of Telematics has developed a framework to close this gap and uses it to demonstrate the performance of data link layer protocols.

This paper presents this framework as well as the requirements and architecture behind it. The current implementation allows remote interaction with OMNeT++ simulations and the presentation of data suitable for demonstrations. It allows for interactive sessions where visitors with little knowledge of the system can modify the simulation and experience the results without an artificial delay. Furthermore, it allows creating simplistic applications with a clean interface designed especially for demonstrations. In contrast to pure OMNeT++ applications, distractions, such as unwanted menus, are removed.

II. State of the Art

This section describes the available features in OMNeT++ for demonstrations and live interaction scenarios. This information is used to derive the need for an additional framework to close the gap between OMNeT++ and our requirements.

A. Data Presentation

Tools for visualising running simulations and their results have been improved over the past OMNeT++ releases. Since version 5.0, 3D visualisation is supported which can be used to display useful information for different scenarios. A feature with a longer history is the graphical view for collected statistics at run-time. While these tools are useful for purposes of debugging and development, it is difficult to use them to present data to other researchers in a demonstration setting. The different graphical views of one simulation do not form a concise user interface that focuses on the important parts while not causing a distraction. Also, it is hardly possible to merge data from different simulations into a single view. While OMNeT++ does support parallel simulation distributed across multiple devices, a feature that is missing is the possibility to simulate on powerful computers while presenting data live on lightweight clients such as tablet computers. A different aspect of performance is related to the data collection at run-time. For live evaluations, OMNeT++ requires results to be stored in vectors, but during long-running demonstrations, these data structures may become too large to handle efficiently.

B. Live Interaction

In most scenarios for OMNeT++ simulations, parameters are provided before simulation start and do not change during execution. In the Tkenv or Qtenv user interface it is, however, possible to drill down to any module and change its parameters at run-time. Using this method requires the affected modules to be designed with this in mind. While OMNeT++ notifies modules when one of their parameters is changed, the module itself has to perform the necessary adoptions under any circumstance. Additionally, OMNeT++ does not offer the possibility to conveniently design a custom GUI to modify the simulation parameters. This is unsuitable for an interactive demo where an uninformed visitor should be able to modify the simulation in predefined degrees of freedom.

III. Requirements for the Framework

For us, the need for a good demonstration platform arose when we wanted to demonstrate the performance of a data link layer protocol [2]. Since our first approaches where hindered by the weaknesses of OMNeT++ as described in the previous section, we decided to collect the requirements for a good simulation control platform for OMNeT++ based projects. A

part of these functional and non-functional requirements for the framework can be derived from a set of use cases which are shown in Fig. 1.

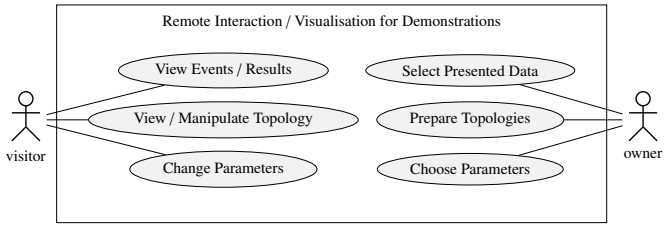


Fig. 1. Main use cases for the demonstration framework

One main goal was the decoupling of visualisation and control from the simulation itself into different applications. Furthermore, it should be possible to separate the execution context of both of these applications so that it is possible to run the simulation on a dedicated machine and visualise it elsewhere. Communication between the OMNeT++ simulation and the front end should be performed using a portable protocol that allows authors to extend the system. This could be used to add new modules that provide or process data and commands such as dedicated hardware platforms or mobile applications. The framework should promote ease of use for both author and visitor of an interactive demonstration. To achieve this, it should be possible to create a simplistic user interface for the visitor while offering a high degree of customisation for the author. All parts of the combined application should be lightweight and efficient to permit long-running and feature-rich demonstrations.

IV. Architecture

The software used for a particular demonstration setup can be partitioned into a front end and a back end with a middleware in between. The front end itself can be composed of one or multiple separate interfaces. In a simple scenario, each interface is a web application displayed in a browser, and each web application itself consists of multiple widgets. Finally, each widgets encapsulates a single feature such as displaying a value, a chart, or a button to send a message.

The back end of the demonstration is formed by OMNeT++ simulations. Here, in contrast to normal simulations, some modules were modified to allow the exchange of messages with the clients. An example of such a module is a special result recorder that collects and publishes a set of statistics. The complete demonstration architecture is depicted in Fig. 2.

A. Middleware

The OMNeT++ simulation and the remote front end exchange commands and data via the Web Application Messaging Protocol (WAMP) [3]. WAMP provides Remote Procedure Calls (RPCs) and publish-subscribe event notifications via WebSockets. While WAMP can be used with either JavaScript Object Notation (JSON) or MessagePack for object serialisation and deserialisation, we have decided to use JSON for the presented framework. Through its nature as a string-oriented

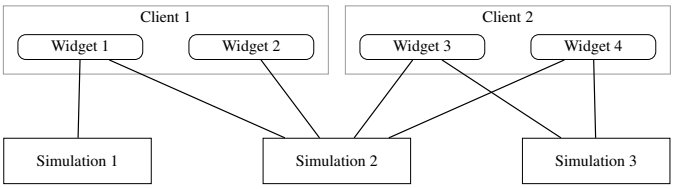


Fig. 2. The architecture with clients connected to a set of simulations. Lines represent connections via the middleware either for RPCs or publish-subscribe.

message format, JSON is not as efficient as the binary MessagePack regarding the size of the serialised data. However, JSON has the advantage of direct support in JavaScript and a human-readable format, which is very useful for the front end development.

Our framework uses RPCs to send control signals from the web front end to the connected OMNeT++ simulations and a publish-subscribe mechanism to direct results back to the GUI. An example of such a control signal would be a reset command that orders one or multiple simulations to restart from the beginning. Results reported via publish-subscribe could be the number of messages lost during the last second or the power consumption per device over a certain period.

B. Back End

The back end part of the framework is developed entirely in *C++* and integrates into the simulation as OMNeT++ modules. To achieve portability, a custom and lightweight implementation of the WAMP protocol is linked into the simulation binary. At the time of development, existing WAMP implementations required a large number of dependencies to be compiled which should be avoided in this case. In the current architecture, every module can be modified to offer RPCs or publishing services by itself. While it is possible to develop generic remote interfaces to OMNeT++ parameters and statistics, this is not required by the architecture so modules can provide RPCs for custom functionality. This flexibility allows the application of this framework to projects of varying complexity and size. To avoid disk space management issues on the simulation side, the back end is only responsible for storing or aggregating a set of statistics until it is published and pushed towards the client. That also means that as long as no front end is connected, no data has to be stored. If the demonstration should show data that covers a time frame of more than a few seconds, the front end is responsible for keeping the required data during the period of use. The simulations in the back end should be able to run for long periods of time without requiring resets or explicit clean-up operations. A relaunch of a front end application is uncritical for the simulation itself because apart from displaying a history, the front end does not have to keep track of a certain state. For simulations, this is different, since some demonstrations might require large amounts of computation time to reach the desired state. Also, long term observation can be an explicit goal of the demonstration.

C. Front End

The client-side software in mind during the development of this framework is a web application built on the widespread technologies of the Hypertext Markup Language (HTML), JavaScript (JS) and Cascading Style Sheets (CSS). Through the open-source WAMP implementation *Autobahn|JS* [4], interaction with the middleware is possible in few lines of code. The combination HTML, JavaScript and CSS is widely used and therefore the barrier to use and extend our framework is kept low. Also, this choice of technology offers the possibility to separate GUI design and application code, and it is supported by a variety of debugging tools. To create a clean interface to one or more OMNeT++ simulations in the back end, the client view is composed of multiple widgets, each with a well-defined responsibility. After the widgets have been developed, the client view is simply composed by instantiating them. If unlocked, widgets can be shown or hidden at run-time and rearranged manually or through switching presets.

The widgets themselves can use *Autobahn|JS* to communicate with the back end while using JavaScript libraries such as *jQuery* [5] or *Chart.js* [6] to display statistics or to interact with the user. Every widget can subscribe to a different publisher if required and the same holds true for the remote procedure calls. A single control element can also call multiple RPCs from different simulations upon a single user interaction. This allows controlling multiple simulations synchronously while combining and comparing their results in a single view. Again, it is possible to develop a generic command multiplexer that relays RPCs towards the corresponding back end, or every widget can take care for that itself. This architecture makes the front end flexible and extensible, and it can thereby be easily adapted for very different scenarios.

V. Implemented Features

With the presented framework, we developed a demonstration of the open source project openDSME [2] [7]. This demo was first presented at the NetSys 2017 [8]. The implementation allows demo visitors to compare the packet delivery ratio and power consumption between two OMNeT++ simulations. One simulation uses basic IEEE 802.15.4 [9] as the data link layer protocol, and the other uses IEEE 802.14.5 DSME. The interface can be used to change the amount of generated traffic as well as the topology by modifying the position of the nodes on the fly. In addition to the current and past values for the packet delivery ratio and the power consumption, the reason for lost transmissions and their location are visualised. A section of the demo interface is shown in Fig. 3.

To illustrate the relation between configuration effort and results, Fig. 4 and the code in Fig. 5 focus on a single widget. This widget communicates with a set of OMNeT++ simulations which each publish the total power consumed during the last simulated second. Once the widget itself has been implemented, using it in a demonstration is simple and requires little configuration. In this case, the widget only expects the ID of an HTML container for placement in the GUI, as well as the addresses and ports of the simulation hosts and the chart labels. The result is shown in Fig. 4.

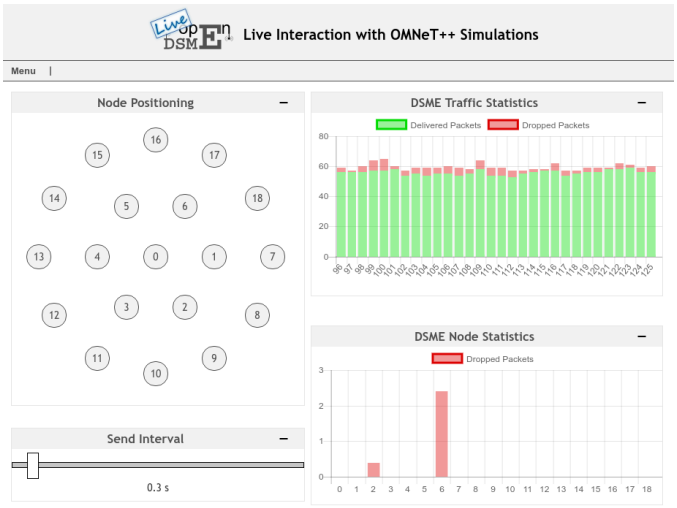


Fig. 3. Screenshot of the live user interface with an oversaturated network. The view in the top left shows the topology; nodes can be dragged to a different position. The bottom left slider allows control over the generated traffic. The right side shows delivered and dropped packets (top) and where exactly packets have been dropped (bottom).

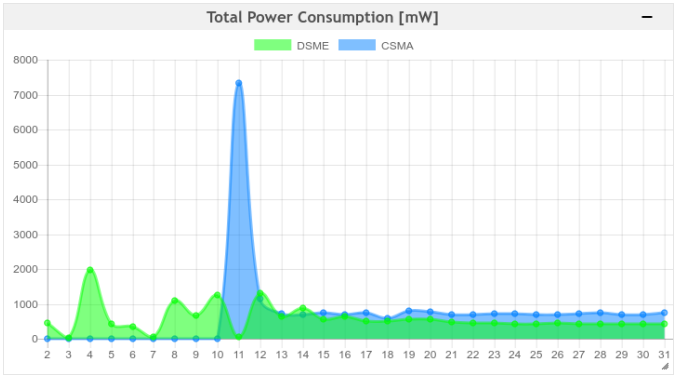


Fig. 4. Screenshot of a widget comparing the power consumption of two different data link layer protocols over time.

A. Available Widgets

Multiple widgets are implemented for our demonstration. The screenshot in Fig. 3 shows a condensed selection of these widgets. Currently, the following collection of widgets is available.

- History of dropped and delivered packets (different colours for queue drop, ACK loss, etc.)
- Location of dropped packets in the last n seconds (different colours for queue drop, ACK loss, etc.)
- Comparison of power consumption between multiple simulations over time
- View on current topology with node drag and drop
- Topology control widget to switch between different predefined topologies

```
<div id="power_chart_container"
     class="draggable ui-widget-content">
  <div class="handle">
    <h2>Total Power Consumption [mW]</h2>
  </div>
</div>

<script>
var dsme_wsuri = "ws://localhost:9002";
var csma_wsuri = "ws://localhost:9003";

var power_statistics
      = new PowerStatisticsModule(
           "power_chart_container",
           [dsme_wsuri, csma_wsuri],
           ["DSME", "CSMA"]
         );
</script>
```

Fig. 5. HTML and JavaScript code required to instantiate the widget shown in Fig. 4.

- Meta widget to enable or disable positioning of widgets as well as exporting and loading views
- Reset button to reset multiple simulations at once
- Traffic slider to control the traffic load for multiple simulations at once by adjusting the average traffic generation interval

One highlighted feature of our framework is the possibility to manipulate multiple simulations simultaneously via a single control element. This requires that all simulations offer a remote procedure with the same name and the same parameters. In our previous demonstration, the only difference between all connected simulations was the data link layer. Since all OMNeT++ instances used the same traffic generator, the only change required for remote control was to add a remote procedure to this module. The client-side JavaScript widget contains a simple slider that represents the mean delay between two packets. This widgets connects to each simulation separately and calls each RPC once the slider changes. On the simulation side, each traffic generator on every node registers a remote procedure to change module parameter that is evaluated every time the next packet generation is scheduled.

VI. Contribution and Future Work

This paper presents the implementation of a framework for remote control and visualisation of live OMNeT++ simulations. The existing possibilities and the requirements for such a system are analysed, and the architecture of our system is presented. We demonstrate the applicability with a demonstration of the openDSME data link layer implementation. The front end code[1] and the implemented OMNeT++ live modules[2] are available at GitHub. We explicitly encourage the reader to take our framework and use it for their demonstrations. Currently, and due to the limited application so far, each OMNeT++ module promotes available statistics and remote procedures by itself. A future modification could be the introduction of more generic modules that offer an interface to parameters without requiring source code additions for each one. Certainly, a more elegant integration into OMNeT++ itself is possible as well to make use of the simulators features and to separate the framework from custom application code. Also, a WAMP dealer service could be employed to further decouple the module and to advertise available simulation data to different front ends. A further application of our framework, which has not been realised yet, would be to introduce other agents and connect them to our front end and back end via the middleware for monitoring and reconfiguring simulation parameters automatically. An example of this would be an application where part of the data comes from simulation and part of it is collected from real hardware implementations, possibly using the FIT IoT-Lab.

[1] https://github.com/openDSME/opplive
[2] https://github.com/openDSME/inet-dsme/tree/opplive/src/opplive

References

[1] András Varga, "The OMNeT++ Discrete Event Simulation System," in *Proceedings of the European Simulation Multiconference (ESM'2001)*, 2001.

[2] M. Köstler, F. Kauer, T. Lübkert, and V. Turau, "Towards an Open Source Implementation of the IEEE 802.15.4 DSME Link Layer," in *Proceedings of the 15. GI/ITG KuVS Fachgespräch Sensornetze*, J. Scholz and A. von Bodisco, Eds. University of Applied Sciences Augsburg, Dept. of Computer Science, Sep. 2016, Technical Reports.

[3] WAMP - The Web Application Messaging Protocol The Web Application Messaging Protocol. [Online]. Available: http://wamp-proto.org/

[4] AutobahnlJS - WAMP for Browsers and NodeJS. [Online]. Available: https://github.com/crossbario/autobahn-js

[5] jQuery. [Online]. Available: https://jquery.com/

[6] Chart.js - Simple yet flexible JavaScript charting for designers & developers. [Online]. Available: http://www.chartjs.org/

[7] Institute of Telematics, TUHH, openDSME. [Online]. Available: http://opendsme.org/

[8] F. Kauer, M. Köstler, T. Lübkert, and V. Turau, "openDSME - A Portable Framework for Reliable Wireless Sensor and Actuator Networks," Demonstration at the International Conference on Networked Systems, Mar. 2017.

[9] *Low-Rate Wireless Personal Area Networks (WPANs)*, IEEE Standard 802.15.4, 2015.

Java Extensions for OMNeT++ 5.0

Henning Puttnies, Peter Danielis, Christian Koch, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany
Tel./Fax: +49 (381) 498-7277 / -1187251
Email: henning.puttnies@uni-rostock.de

***Abstract*—On the one side, network simulation frameworks are important tools for research and development activities to evaluate novel approaches in a time- and cost-efficient way. On the other side, Java as a highly platform-independent programming language is ideally suited for rapid prototyping in heterogeneous scenarios. Consequently, Java simulation frameworks could be used to firstly perform functional verification of new approaches (and protocols) in a simulation environment and afterwards, to evaluate these approaches in real testbeds using prototype Java implementations. Finally, the simulation models can be refined using real world measurement data. Unfortunately, there is to the best of our knowledge no satisfying Java framework for network simulation, as the OMNeT++ Java support ended with OMNeT++ version 4.6. Hence, our contributions are as follows: we present Java extensions for OMNeT++ 5.0 that enable the execution of Java simulation models and give a detailed explanation of the working principles of the OMNeT++ Java extensions that are based on Java Native Interface. We conduct several case studies to evaluate the concept of Java extensions for OMNeT++. Most importantly, we show that the combined use of Java simulation models and C++ models (e.g., from the INET framework) is possible.**

I. Introduction

Network simulators are perfectly suitable for the early evaluation of innovative approaches (e.g., novel applications or protocols for communication, control, or security). Popular simulators like OMNeT++ are recommendable as they offer many existing modules suitable for reuse, generally have a good usability, and are seriously tested. Furthermore, the Java programming language is perfectly suitable for rapid prototyping as it is very predictable, easy to debug, and highly platform independent. As a consequence, Java programs can be deployed on different platforms with minimal adaptation effort, which enables the evaluation of an approach on many different devices in heterogeneous IoT scenarios.

Therefore, it is an interesting approach to combine OMNeT++ and Java like it was possible using the Java extensions for the OMNeT++ Versions 3.X to 4.6. Firstly, using Java and OMNeT++ enables to evaluate an approach using simulation models written in Java. Secondly, the Java simulation models can be used to develop a platform independent Java prototype implementation and thus, quickly evaluate research approaches in real world scenarios. As a result it is possible to feed the results (e.g., realistic computation times) back into the simulation models. Our contributions are as follows:

- We generated and present Java extensions for OMNeT++ 5.0 derived from the existing Java extensions for OMNeT++ 4.6 and give a brief overview of the Java capabilities of other existing simulation frameworks.
- We explain in detail how the Java extensions work in combination with OMNeT++. In contrast to the existing documentation of the Java extensions that focus on how to use the Java extensions, we turn our attention to their functional principles. We want to help other researchers in understanding the Java extensions for OMNeT++ and encourage them to use Java simulation models in OMNeT++.
- We conduct several case studies and evaluate the effort of porting a real Java application to a Java simulation model. Moreover, we present the possibility to combine Java simulation models in OMNeT++ with existing C++ models (e.g., the INET library). This is an important case study, as the reuse of existing modules can tremendously reduce the time to develop a new simulation model. To the best of our knowledge, this is the first analysis and proof of concept implementation of the combination of Java simulation modules and existing C++ modules (e.g., INET framework), besides the use of the OMNeT++ simulation kernel.
- The entire system including all source code is freely available for download. We share a virtual machine (VM) for VirtualBox [1] running Ubuntu. Therefore, the system is running "out of the box" without any platform dependencies (e.g., compilers, Java Virtual Machine) or the need for configuration (e.g., paths, environment). This is an important point as the OMNeT++ Java extensions base on Java Native Interface (JNI), which highly depends on the environment (operating system, Java virtual machine, paths etc.). Thus, sharing a VM facilitates the use of the Java extensions.

II. Related Work

As the generation of Java extensions (interfaces) for a C++ simulation framework (like OMNeT++) requires substantial efforts, we analyze the Java capabilities of existing simulators in the following.

The Network Simulator 3 (NS-3) [2], [3] is a very popular simulation framework for computer networks and the successor of NS-2, although NS-3 was developed from scratch. It

is written in C++ and to the best of our knowledge, there is at the time of writing no existing approach to integrate Java simulation models into NS-3.

Java Network Simulator (JNS) [4] is a Java implementation of NS-2. Nevertheless, the development stopped in 2010 and it always had less features than NS-2 as stated by the developers.

JNetworkSim [5] is a modular and fast simulator developed at Stanford University. Nevertheless, it focuses on the simulation of network switches rather than entire network topologies. Furthermore, it was lastly updated in 2007.

Another Java simulation framework is the Probabilistic Wireless Network Simulator (JProwler) [6] that is a simple wireless network simulator. It is (as stated by the developers) small and lightweight. However, as the download website has not been updated since 2008, we assume that it is no longer maintained.

Psimulator2 [7] is a basic graphical network simulator originally developed at the Czech Technical University in Prague. Although it is still under maintenance, the purpose of this simulation framework is the teaching of basic networking topics. In contrast, we focus on research activities as motivated in the introduction.

Furthermore, there is a free network simulator written in Java [8], which has the ability to reconfigure routing and topology. It is stated to be cycle-based and to model flit-level communication. Assumingly, it is out of maintenance as the initial author James Hanlon finished his Ph.D. in 2014.

After analyzing related works, we can conclude that there is no existing simulation framework combining a popular simulator (good for reuse) and the ability to development simulation models in Java (good for rapid prototyping). We address this need with the Java extensions for OMNeT++ 5.0.

III. Concept of Java Extensions for OMNeT++

A. Working Principles of Java Extensions for OMNeT++

The Java extensions for OMNeT++ base on the JNI [9]. To allow the use of Java simulation models, a new simulation executable is generated. This simulation executable (jsimple) consists of the entire OMNeT++ simulation kernel as well as several extension modules. As depicted in the class diagrams (see Fig.1), the developer can write a Java simulation model (e.g., *MyModule.java*) that inherits from *JSimpleModule.java*, which is a Java wrapper for the C++ class *JSimpleModule.cc*. *JSimpleModule.cc* is an extension class for OMNeT++ that implements the interface between the Java simulation modules (e.g., *MyModule.java*) and the OMNeT++ simulation kernel (via *cSimpleModule.cc*).

Fig. 4 depicts the flow chart of an exemplary execution of OMNeT++ with Java extensions. The program *jsimple.exe* is started as simulation executable and reads the corresponding **.ini* file (e.g. *MySim.ini)*. Let us assume, that *MySim.ini* loads a **.ned* file that uses *MyModel* (a Java simulation model) from Fig.1. The *JSimpleModule::initialize()* method is calling the *JUtil::initJVM()* method to start the Java virtual machine (JVM). As the JVM is a shared library that can execute Java

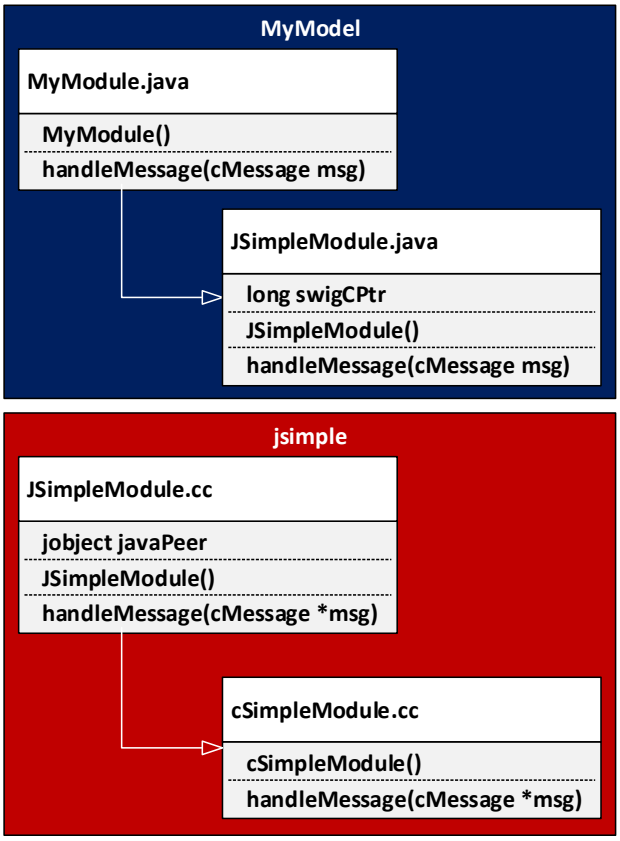


Fig. 1. Class diagrams of extension modules (e.g., JSimpleModule) that form the Java extensions. Blue: Java code, Red: C++ code. *JSimpleModule.java* is a wrapper for *JSimpleModule.cc*. Therefore, it has the member *long swigCPtr* that is a pointer to *JSimpleModule.cc*. Vice versa, *JSimpleModule.cc* has the member *jobject javaPeer* that is a pointer to *JSimpleModule.java*.

bytecode in **.class* files, it is possible to execute Java simulation models. In the Java simulation model, the constructor of *MyModule* is called that in turn calls the constructor of *JSimpleModule* and C++ code via JNI. It is possible to call a Java method from C++ code and vice versa. Figures 2 and 3 depict and explain exemplary code snippets of the JNI calls.

```
jenv->CallVoidMethod(javaPeer, doHandleMessageMethod);
```

Fig. 2. Calling a Java method from C++ (*JSimpleModule::handleMessage()*): jenv = pointer to java environment; CallVoidMethod = calls a method of an object; javaPeer = pointer to the java peer of this object (in this case: *JSimpleModule.java*); doHandleMessageMethod = method ID of *handleMessage()*

```
super(SimkernelJNI.SWIGJSimpleModuleUpcast(cPtr), false);
```

Fig. 3. Calling a C++ method from Java (constructor of *JSimpleModule.java*): super = constructor of ancestor (*cSimpleModule*); SimkernelJNI = class holding Java wrappers for all native (C++) methods; cPtr = pointer to corresponding C++ class (*JSimpleModule.cc*); false = dummy value

The *SimkernelJNI_registerNatives()* method is used in original Java Extensions and the Java Extensions for OMNeT++ 5.0. A PERL script provided by the OMNeT++ 4.6 (*reg-*

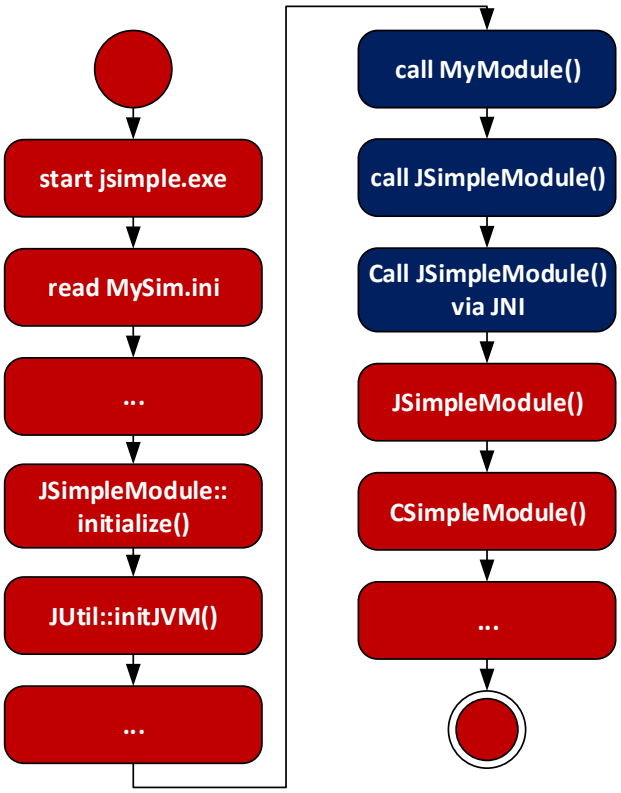


Fig. 4. Flow chart of the OMNeT++ with Java extensions executing a Java simulation model. Blue: Java code, Red: C++ code

isternatives.pl) automatically generates this method. *SimkernelJNI_registerNatives()* registers all C++ methods that are wrapped and accessible from Java code. The purpose of the *SimkernelJNI_registerNatives()* method is to check the signature of every C++ method that is wrapped and accessible from Java code, before this method is actually used [10]. This substantially increases the stability of the system, as there are more than 1000 methods accessible from Java code and otherwise the signatures are only checked at runtime, which might lead to unstable behaviour.

B. Generating the Java Extensions

In what follows, we will describe the process of generating the Java extensions. The same approach might be used to generate wrappers for any C++ simulation models to reuse them in conjunction with Java simulation models. The Java wrapper classes for OMNeT++ are automatically generated using SWIG [11], [12], which is a well-documented and powerful tool that can be used to generate an interface between ANSI C/C++ and many other high level languages (e.g., Java, Chicken, and Python). SWIG uses the JNI API for this interface. Files that are of the type **.i* (interface files) are used to configure the process of generating the interface correctly. There are several specifics when using SWIG and generating Java Extensions for C++ code (e.g., OMNeT++). Therefore, we focus on the most important ones in the following.

The overloading of operators is available in C++ but not in Java. As a consequence, these operators are wrapped into methods (e.g. "=" is wrapped into *set()*, "==" into *sameAs()*, and "++" into *incr()*). This can be done automatically by applying the *%rename* directive in the SWIG interface file.

As there are no pointers available in Java, specific SWIG pointers exist, which are wrappers holding the address of the corresponding C++ object as a Java long.

SWIG can handle namespaces but ignores them in the names of the Java wrapper code. Therefore, a method *Foo::Bar()* in C++ is wrapped into a method *Bar()* in Java by SWIG. If two methods ,which have the same name, are defined in two separate namespaces (e.g., *Foo1::Bar()* and *Foo2::Bar()*), SWIG maps them to the same Java method name (e.g., *Bar()*) that is therefore defined multiple times and the compile process crashes. The solution is to use the *%rename* directive to rename *Foo1::Bar()* into *Foo1_Bar()*.

SWIG does not support (and hence ignores) nested classes (class definitions in classes). The solution consists in a redefinition of the inner class in the SWIG interface file (**.i*). This redefinition makes the inner class globally visible and hence SWIG generates a wrapper for this class.

C. Differences between the Java Extensions for OMNeT++ 4.6 and OMNeT++ 5.0

As the reuse of existing software is a powerful method to speed up the development process and improve its results, we have not developed the Java extensions for OMNeT++ 5.0 from scratch. In constrast, our Java extensions are derived from the Java extensions for OMNeT++ 4.6. In the following, we describe the difference between the Java extensions for OMNeT++ 4.6 and the Java extensions for OMNeT++ 5.0.

Whereas there are only small differences between the Java extensions for the OMNeT++ versions between 4.1 and 4.6 (two additional Lines for cCompoundModule in the SWIG interface file), we apply multiple changes:

In the SWIG interface file, we have to include all OMNeT++ headers explicitly (e.g., *simkerneldefs.h* → *omnetpp\simkerneldefs.h*), as SWIG does not follow C++ includes. As described previously, SWIG is able to handle namespaces. Consequently, we have to refer to all C++ classes with the namespace prefix (*omnetpp::*) in the SWIG interface file. All generated Java wrapper classes are referred to without a prefix. Furthermore, we removed several header files (e.g., *cevent.h* and *ceventheap.h*) from the SWIG interface file and added a few (e.g., *cmessageheap.h*).

In the extension classes (JSimpleModule, JMessage, JUtil), we moved all declarations and definitions into the namespace omnetpp to cope with this namespace introduced by OMNeT++ 5.0. This allows a dynamic name solving within the source code of the C++ classes (e.g., *JSimpleModule.cc*) similar to the Java extensions for OMNeT++ 4.6.

The *registernatives()* method now registers a total number of 1926 Java wrapper methods. In comparison, 1400 Java wrapper methods where available in the Java extensions for OMNeT++ 4.6.

IV. Case Studies: Java Simulation Models in OMNeT++ 5.0

Firstly, we used the Jsamples project that is provided with the Java Extensions for OMNeT++ 4.6. The Jsamples project consists of several sample applications (TicToc etc.) that serve as tutorial similarly to their C++ counterparts. We used this project to test our Java extensions for OMNeT++ 5.0 successfully.

A. Converting a Java UDP Ping Implementation to a Simulation model

As first cast study, we developed a UDP-based Ping implementation in Java to evaluate the effort for porting a real Java application to a Java simulation model. Firstly, we tested the application in a real network consisting of two Galileo uno boards [13]. Secondly, we converted this real Java application to a Java simulation model that is executable using OMNeT++ 5.0 with Java extensions. For the functionality of the Java application, the porting process turned out to be easy. However, regarding the API for the communication interface (in this case: UDP sockets), the Java code has to be adapted to the working principles of OMNeT++ (especially communication based on *cMessages*). This is a noticeable effort as the API of system calls (e.g., UDP sockets) and the API of OMNeT++ (or INET) are completely different.

B. Using the INET Framework in Conjunction with Java Simulation Models

As second case study, we evaluated the interface between the modules of the INET framework and Java simulation models. This is an important case study as the reuse of existing INET modules is an outstanding way to speed up the development of new simulation models. As first step, we executed the INET wireless tutorial that entirely consists of C++ modules using the jsimple simulation executable (simulation kernel and Java extensions). Hereby, we showed that the INET modules can be used in **.ned* files and executed by jsimple. As second step, we developed a simulation model consisting of both Java and C++ modules to analyze whether they can be connected and communicate with each other. Our simulation model (see Fig.5) is derived from the INET example *"inet/examples/ethernet/lans/twoHosts.ini"*. We used the *EtherHost* from INET and connected it with our own implementation called *myEtherHost*. The module *myEtherHost* consists of *EtherLLC*, *EtherQoSQueue*, and *IEtherMAC* from INET written in C++ and *EtherEchoSrv*, which is a Java module. The evaluation showed, that *EtherEchoSrv* (Java) can correctly register to *EtherLLC* (C++) using the *Ieee802Ctrl* (C++) control structure form INET. Moreover, the module *EtherEchoSrv* (Java) correctly receives Ethernet packets from the INET module and sends back valid Ethernet packets, which are accepted by the *EtherLLC* and *IEtherMAC* (both C++) modules from INET. Thus, we showed that even the combination of INET modules and Java simulation modules is possible.

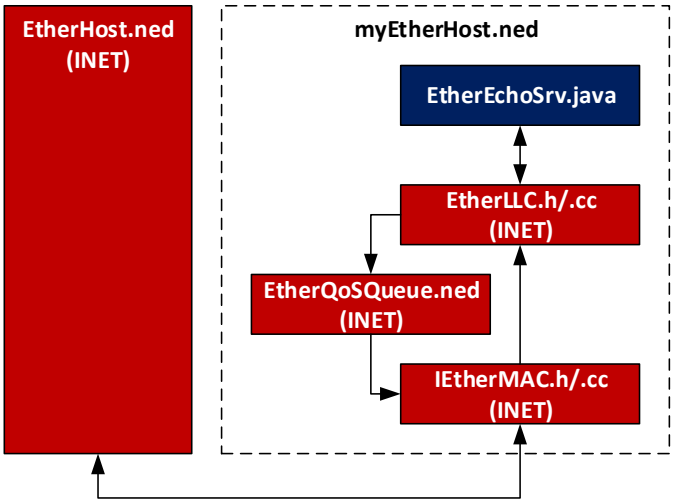


Fig. 5. Overview of a simulation model that consists of both, Java simulation modules and existing C++ modules (here: from the INET framework). Blue: Java code, Red: INET code (C++ and **.ned*).

C. Limitations of the OMNeT++ Java Extensions

Although the porting of a simulation model to a real application is straight forward for simple programs, there are several limitations of OMNeT++ and the Java extensions.

Multithreading is not supported within one application. Therefore, extensive implementations using multiple threads have to be reduced into one thread or partitioned into several LPs (logical processes). This however is a general disadvantage of OMNeT++ [14] and thus does not only apply to the Java extensions.

Moreover, the modelling of concurrent behaviour using while loops is not adequate, because the simulation would freeze if an operation is not atomic. It is impossible to send a packet and receive the response within one execution of the *handleMessage()* method, while a similar behaviour is possible within the *main()* method of a Java application. This might necessitate manual modifications of the Java code. Nevertheless, this is not a drawback introduced by the Java extensions.

V. Summary and Outlook

In this paper, we presented the Java extensions for OMNeT++ 5.0 and describe the generation of the OMNeT++ Java extensions as well as their functional principles. Furthermore, we conducted several case studies to evaluate the effort of porting real Java applications to Java simulation modules. Although our Java extensions are derived from the existing Java extensions for OMNeT++ 4.6, we enable the possibility to interface Java simulation modules and C++ modules (e.g. from the INET library). This was not examined before but is important for the reuse of existing modules and therefore for the utility of the Java extensions.

The generation of Java extensions for OMNeT++ 5.1 would be of interest for future work. This should be easier than the generation of the extensions for OMNeT++ 5.0 as the

difference between OMNeT++ 5.0 and OMNeT++ 5.1 is much smaller than the changes between OMNeT++ 4.6 and OMNeT++ 5.0.

VI. Downloading the Source code

The entire system (OMNeT++ 5.0, Java extensions, and case studies) is available online[1] as an Ubuntu virtual machine for VirtualBox. In contrast to all previous versions of the Java extensions for OMNeT++, there is neither a need to regenerate the Java extensions nor to recompile the jSimple simulation executable (OMNeT++ simulation kernel with Java extensions). This is an enormous benefit as JNI-based systems are highly dependent on platforms and paths.

References

[1] "Oracle virtualbox." [Online]. Available: https://www.virtualbox.org/
[2] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, 2008.
[3] "Network simulator 3." [Online]. Available: https://www.nsnam.org/
[4] "Java network simulator." [Online]. Available: http://jns.sourceforge.net/
[5] "Jnetworksim." [Online]. Available: http://yuba.stanford.edu/JNetworkSim/
[6] "Jprowler." [Online]. Available: http://www.isis.vanderbilt.edu/projects/nest/prowler/
[7] "Psimulator2." [Online]. Available: https://github.com/rkuebert/psimulator
[8] "Free java network simulator." [Online]. Available: http://www.java2s.com/Open-Source/Java_Free_Code/Network/Download_network_simulator_Free_Java_Code.htm
[9] "Java native interface." [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/functions.html
[10] "Android java native interface tips." [Online]. Available: https://developer.android.com/training/articles/perf-jni.html
[11] D. M. Beazley *et al.*, "Swig: An easy to use tool for integrating scripting languages with c and c++," in *Tcl/Tk Workshop*, 1996.
[12] "Simplified wrapper and interface generator (swig)." [Online]. Available: http://www.swig.org/
[13] "Intel galileo board." [Online]. Available: https://www.arduino.cc/en/ArduinoCertified/IntelGalileo
[14] "Omnet++ simulation manual: Section 16 parallel distributed simulation." [Online]. Available: https://omnetpp.org/doc/omnetpp/manual/#cha:parallel-exec

[1] https://bwsyncandshare.kit.edu/dl/fi8R6skmuBPh6UfXHWzcgBxt/.zip