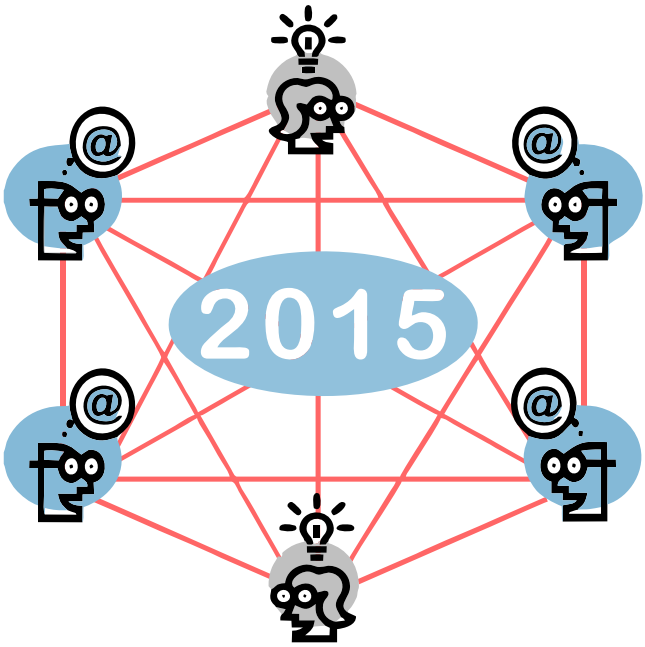
OMNeT++ Community Summit 2015

Successor of the International Workshop on OMNeT++

IBM Research – Zurich, Switzerland — September 3-4, 2015



Proceedings of the 2nd OMNeT++ Community Summit

Community Summit Organizers

Founding Chair

Andras Varga (Simulcraft Inc.)

Community Summit Organizers

Anna Förster (University of Bremen, Germany)

Cyriel Minkenberg (IBM Research - Zurich, Switzerland)

Technical Program Organizers

German Rodriguez Herrera (IBM Research - Zurich, Switzerland)

Michael Kirsche (BTU Cottbus–Senftenberg, Germany)

Publicity Organizer

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

Technical Program Committee

James Brusey (University of Coventry, UK)

David Eckhoff (University of Erlangen-Nürnberg, Germany)

Laura Marie Feeney (SICS, Schweden)

Michael Frey (HU Berlin, Germany)

Yutaka Matsubara (Nagoya University, Japan)

Alfonso Ariza Quintana (University of Malaga, Spain)

Christoph Sommer (University of Paderborn, Germany)

Mirko Stoffers (RWTH Aachen University, Germany)

Antonio Virdis (University of Pisa, Italy)

Matthias Wählisch (FU Berlin, Germany)

Welcome Note from the OMNeT++ Community Summit Organizers

The second edition of the OMNeT++ Community Summit took place in September 2015 at IBM Research - Zurich in Switzerland.

The aim of the OMNeT++ Community Summit is to provide a forum for discussions on recent developments and novel ideas in the broad area of network simulation and modeling, with a focus on the OMNeT++ simulation environment. After six successful editions of the ACM/ICST International Workshop on OMNeT++ that started back in 2008, we decided to switch from the standard scientific workshop format to a new and open (access) format. This new format allows us to better encompass more visionary ideas and foster interaction between the participants. The first Summit in 2014 in Hamburg, Germany, was a huge success with over 40 on-site and several remote participants. In 2015, we continued to expand the idea of the community summit by explicitly addressing the interaction through tutorials, discussion panels and an extension of the summit over two days. We also provided a peer-reviewing process to strengthen 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, to name a few, on Internet protocols, wireless networks, overlay networks, and vehicular networks. 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.

If we look at the technical program, we can see that the second edition of the summit continued to exceed our expectations: researchers and developers from over 10 countries, from industry and universities and research institutes alike, responded to our call for participation and submitted their work. We received a wide range of submissions with propositions of new simulation models, extensions of OMNeT++ and its ecosystem, but also tutorials and panel discussions about the future of OMNeT++ and modeling itself. The discussed topics range from modeling and protocol design on all layers, application fields like wireless sensor networks and quantum cryptography, to the interconnection of standalone simulation models and frameworks. We have gathered all individual contributions and panel submissions of the 2015 summit in the form of proceedings to enable future reference and further discussion within the community.

We want to thank our sponsor - IBM Research Zurich - for providing outstanding financial and logistic support, which allowed us to omit conference fees and hold the summit in the great facilities of IBM in Zurich. We thank our Publicity Organizer, Kyeong Soo (Joseph) Kim (Xi'an Jiaotong-Liverpool University, China) for his excellent job in presenting the OMNeT++ Summit to the community. Last but certainly not least: we thank all participants of the OMNeT++ Community Summit 2015 for their work and attendance. You are essential for the success of the Community Summit.

Anna Förster (University of Bremen)
Cyriel Minkenberg (IBM Research - Zurich)
Community Summit Organizers

German Rodriguez Herrera (IBM Research - Zurich)
Michael Kirsche (BTU Cottbus-Senftenberg)
Technical Program Organizers

Contents

SESSION - POSTERS AND DEMONSTRATIONS

SESSION - FUTURE OF OMNeT++

SESSION - FUTURE OF INET

PANEL - ENERGY CONSUMPTION MODELING

MiXiM, PAWiS, and STEAM-Sim Integration – Combining Channel Models, Energy Awareness, and Real-life Application Code

Georg Möstl* and Andreas Springer‡
*Institute for Integrated Circuits, Johannes Kepler University Linz, georg.moestl@jku.at
‡Institute for Communications and RF-Systems, Johannes Kepler University Linz, a.springer@nthfs.jku.at

***Abstract*—After a decade of research in the field of wireless sensor networks (WSNs) there are still open issues. WSNs impose several severe requirements regarding energy consumption, processing capabilities, mobility, and robustness of wireless transmissions. Simulation has shown to be the most cost-efficient approach for evaluation of WSNs, thus a number of simulators are available. Unfortunately, these simulation environments typically consider WSNs from a special point of view. In this work we present the integration of three such specialized frameworks, namely MiXiM, PAWiS, and STEAM-Sim. This integration combines the strengths of the single frameworks such as realistic channel models, mobility patterns, accurate energy models, and inclusion of real-life application code. The result is a new simulation environment which enables a more general consideration of WSNs. We implemented and verified our proposed concept by means of static and mobile scenarios. As the presented results show, the combined framework gives the same results regarding the functionality and energy consumption as our "golden model". Therefore the system integration was successful and the framework is ready to be used by the community[1].**

I. INTRODUCTION

A network consisting of tiny, processing ressource limited, and battery powered sensor nodes which communicate wireless is called a wireless sensor network (WSN). The nodes monitor environmental quantities like tension, pressure, sound, temperature, and acceleration. WSNs are used in home automation, car-interior devices, container monitoring and tracking, health-related deployments, and military applications. Some of these applications require highly mobile nodes and therefore mobility comes into play. Further, the lifetime of a sensor network and thus the energy efficiency is a crucial requirement. Typically a node is expected to "live" several years powered by a coin cell. In an increasingly wireless world WSNs are used in industrial applications which imply closed-loop systems. The class of WSN is extended to the class of wireless sensor and actuator networks (WSANs). Therefore the time behavior of software and hardware components becomes essential to guarantee realtime constraints. Another quantity of main interest is the reliability of the network which is dominated by the behavior of the wireless channel.

We conclude that a feasible and accurate simulation of WSNs and WSANs must include mobility of the nodes, sophisticated channel models, fine granular modeled timing, execution of real-life application code, and energy awareness. Since the modeling and simulation of WSNs and WSANs is a complex task and requires an in-depth knowledge of various components, it is best practice to use existing simulation frameworks.

The MiXiM [1] simulation framework focuses on node mobility and accurate channel models. MiXiM itself is a combination of three simulation frameworks. The Mobility Framework provides the general structure, mobility, and connection management for the framework. Radio propagation models are included from the Channel Simulator. The models available in the Positif framework, MAC Simulator, and the Mobility Framework are used as protocol libraries. Another simulation framework is the PAWiS [4] framework which focuses especially on accurate simulation of the energy consumption of a network. With PAWiS it is possible to establish an electrical network consisting of power suppliers and consumers, which influence each other. Further, PAWiS provides a clear separation of software and hardware components and enables the modeling of arbitrary hardware such as a radio transceiver, CPU, or a sensor interface. STEAM-Sim [2] is a recently published simulation environment which enables the simulation of the timing and functionality of real-life firmware, i.e., the software executed by a nodes CPU. The so called *time annotation engine* parses arbitrary firmware code written in the high-level language C and determines the execution time of code blocks. The annotated code is afterwards combined with hardware models developed using PAWiS and simulated.

To get the best of each simulation framework we combine them into a new simulator:

- MiXiM – realistic channel models and mobility
- PAWiS – sophisticated energy models and modeling of arbitrary hardware
- STEAM-Sim – timing and functionality of real-life firmware

The overall simulation environment setup is shown in Fig. 1 which builds up on the discrete event simulator OMNeT++ [3]. As STEAM-Sim already uses hardware models developed in PAWiS, in this work we concentrate more on the discussion of the integration of the MiXiM and the PAWiS framework. As an outcome, it is possible to establish a simulation which

[1]available for download at *http://sourceforge.net/projects/steamsim*

provides accurate timing of hardware and software components, accurate energy consumption behavior, realistic channel models, and mobility of WSNs and WSANs.

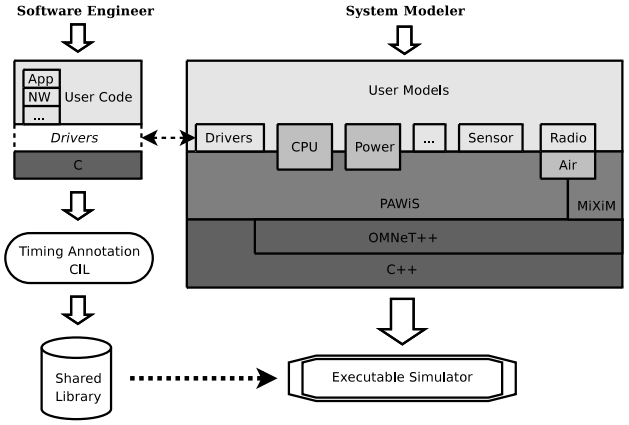


Fig. 1. Developed system integration of MiXiM, PAWiS, and STEAM-Sim

II. Implementation

In this section we discuss the integration of the MiXiM [1] and the PAWiS [4] simulation frameworks. This is a complex task as MiXiM implements a layered view of WSNs complying to the ISO/OSI model, whereas PAWiS provides a hardware/software partitioning of WSNs.

A. Resolving version conflicts

The first step was to solve the version conflicts between PAWiS and MiXiM, which prohibits a combined usage of the frameworks. Both frameworks are based on different versions of OMNeT++. PAWiS requires v3.2, whereas MiXiM requires v4.1 or higher. Unfortunately, there are major differences between versions 3.x and 4.x like the internal representation of the simulation time.

To keep up with the development of OMNeT++ we decided to port PAWiS to OMNeT++ v4.1. Although there are some scripts available to automatically port OMNeT++ simulation models from v3.2 to v4.1 we had to do some manual fine-tuning. For example the interfaces of the `sendDirect` and `sendDelayed` methods changed.

B. Integration of MiXiM and PAWiS

The integration was accomplished in three steps: (i) identifying how to map the functionality of MiXiM with PAWiS, (ii) determining the interfaces and deriving the inheritance of the developed modules, and (iii) establishing the interconnection in the corresponding NED files.

1) Functionality Mapping: Fig. 2(a) depicts the mapping of the functionality between MiXiM and PAWiS for transmissions. As can be seen on the left-hand side in Fig. 2(a), a transmission is originated in PAWiS calling the `sendToAir()` method of an `AirClientModule` object. The `msg` parameter points to a PAWiS `AirMessage` which comprisis data to be transmitted and some control information. The message is encapsulated in a new MiXiM MAC packet (`MacPkt`) and is sent to the MiXiM physical layer (Phy). Therefore the `AirClientModule` is interpreted as the MAC layer from the MiXiM point of view. In the Phy the packet is handled as a MAC packet and (i) is sent to the channel and (ii) results in a scheduled self message marking the end of the transmission. This TX_OVER message is encapsulated into a PAWiS `AirMessage` and is sent to the `AirClientModule` object. When the object receives the message, `onAirDataTransmitted()` is invoked and the transmission is completed at the sender.

As depicted in Fig. 2(b) an incoming packet is first handled in the MiXiM Phy by the `handleAirFrameStartRx()` method. This marks the start of the reception. Subsequently, two tasks have to be accomplished. Firstly, a new self message is scheduled in the Phy, which marks the end of the reception. Secondly, the MiXiM *decider* is invoked to process the new *signal*, thus a new PAWiS `AirMessage` is created and is sent to the PAWiS air client. This message is handled in the `handleMessage()` method. Subsequently, it is evaluated if the transceiver is ready to receive this message via a call to `acceptAirDataStart()`. Additionally, the interfering noise from neighbor channels is calculated executing `calcInterferingNoise()`. If the packet is accepted by the radio transceiver, the PAWiS preview mechanism is invoked (`onPreviewPacket()`) which allows to split the packet reception into interesting parts such as receiving the preamble, the synchronization word, the payload, etc. A change in the SNIR results in a call of the PAWiS `calcBitErrors()` method.

When the last byte of the packet has been received, the corresponding `handleAirFrameEnd()` method is called in the MiXiM Phy, which subsequently calls the `processSignalEnd()` method in the decider. Afterwards the received MAC packet is converted to a PAWiS `AirMessage` and is sent upwards to the PAWiS framework where the signal/noise level is updated accordingly. Finally, `onAirDataArrived()` is invoked in the `AirClientModule` which marks the end of the reception.

2) Structure, Interfaces and Inheritance: Fig. 3 depicts the necessary modules for integrating MiXiM and PAWiS in a simulation run. Modules highlighted in grey are mandatory running a combined simulation.

From the point of view of PAWiS there must be an instance of the `MiximAirClientModule` class which represents a model of a radio transceiver such as a CC2500. This `MiximAirClientModule` is derived from the PAWiS internal `AirClientModule` class. As can be seen in Fig. 3 this external transceiver is connected to a microcontroller via a serial interface (SPI). This interface is used by the firmware running on the microcontroller to configure and control the transceiver. Three methods in the `MiximAirClientModule` had to be developed. Firstly, the `onStartup()` method of every PAWiS module is invoked during initialization of the simulation. In this method OMNeT++ gates (`upperGate` and `upperCtrlGate`) and

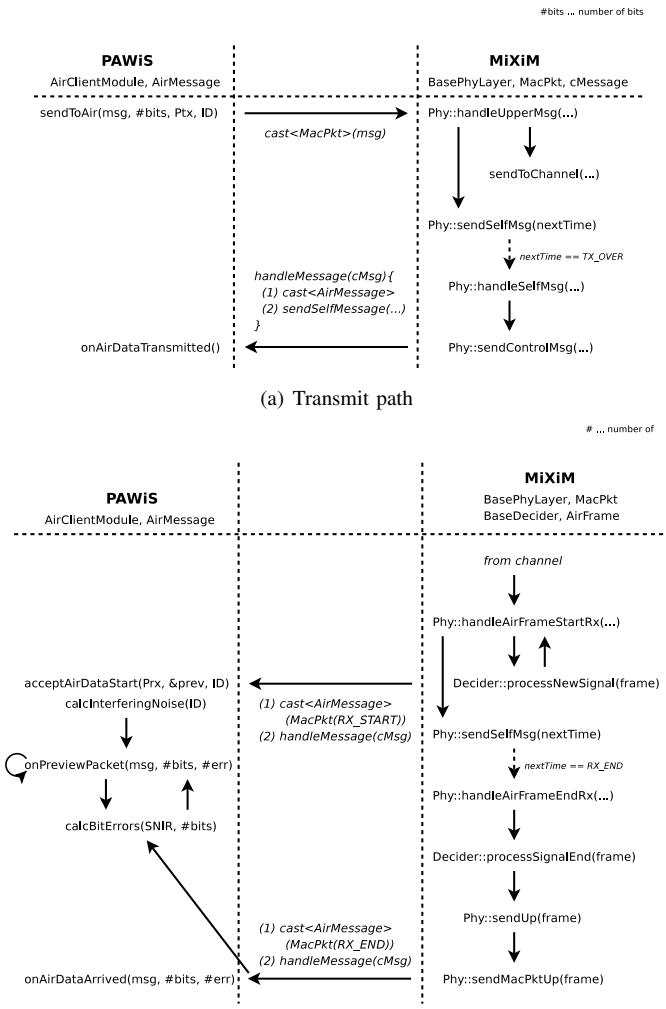


Fig. 2. MiXiM and PAWiS integration – mapping functionality for the transmit and receive operations

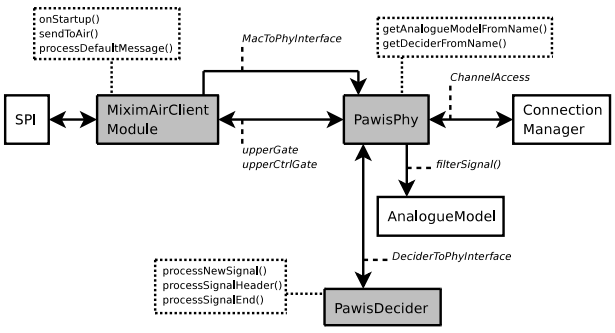


Fig. 3. MiXiM and PAWiS integration – structure and interfaces (modules highlighted in gray are mandatory)

a reference to the MiXiM Phy (`MacToPhyInterface`) are retrieved and locally stored. This ensures efficient handling of further interactions with the MiXiM Phy. Subsequently, the Phy is set to the receiving state. Secondly, the `sendToAir()` method is used to encapsulate a PAWiS `AirMessage` into a MiXiM `MacPkt` (including a MiXiM signal). Thirdly, `processDefaultMessage()` is invoked by the `handleMessage()` method and processes the reception of a MiXiM `MacPkt` sent from the Phy to the PAWiS radio transceiver. Depending on the type of message the corresponding actions are invoked such as an update of the SNIR in PAWiS.

In MiXiM we introduced two modules namely `PawisPhy` and `PawisDecider`. The `PawisPhy` class is derived from the MiXiM `PhyLayer` and implements methods to initialize the used MiXiM *analogue model* and the decider from NED parameters (`getAnalogueModelFromName()` and `getDeciderFromName()`). The `PawisDecider` class which is a subclass of the MiXiM `BaseDecider` is controlled by the Phy using the `DeciderToPhyInterface`. Both, the Phy and the decider have internal references to each other, therefore no OMNeT++ message passing is used. Three methods of the decider were overwritten. First, the `processNewSignal()` method is invoked when a reception starts and implements the cast of a MiXiM `AirFrame` to a message which can be handled by the PAWiS transceiver model. The `PawisDecider` was designed in such a way that it is possible to handle multiple concurrent signal receptions. Second, `processSignalHeader()` is just an empty stub and is left open for further development. Third, the `processSignalEnd()` method implements the indication of the end of reception to the PAWiS `MiximAirClientModule` object.

3) NED Structure: MiXiM defines a fixed structure of the network setup and the basic nodes. This structure is defined in OMNeT++ NED files. At the top level of a MiXiM simulation model there is the `BaseNetwork` which, for example, defines the used connection manager. To integrate PAWiS into the simulation the `BaseNetwork` instantiates two PAWiS models namely the `PawisLogRS232` for logging a nodes serial output and the `Config` module needed for configuration.

A node in MiXiM must be derived from type `BaseNode` where a mandatory mobility module, an ARP module, and a utility module (e.g., blackboard) are instantiated. To integrate PAWiS, there is only one change necessary, i.e., the definition of a unique node identifier `MiximNodeId`. As a result of the integration process, within a node there is only the `nic` (network interface) left. The network and application layer are left empty because they are linked into the simulation by means of the PAWiS node.

III. Experiments and Results

We evaluated the successfull integration of MiXiM, PAWiS, and STEAM-Sim investigating a real-world received signal strength indicator (RSSI) readout scenario. A base station transmits a synchronization frame (beacon) every second to

trigger several sensor nodes to deliver data. The base station receives the data of the sensor nodes and logs the RSSI values of correctly received packets (CRC check passed) to a serial interface. Wireless transmissions are separated in time and therefore a time division multiple access (TDMA) scheme is implemented. For the purpose of simulation the output to the serial interface is saved in a log-file.

Each node comprises an MSP430FG4618 microcontroller and a CC2500 transceiver to transmit a packet consisting of 4 bytes of preamble, a 4-byte synchronization word, 3 bytes of header (length, address, packet type), 2 bytes payload and 2 bytes CRC using 2-FSK modulation at a datarate of 2.4 kBaud and an output power of 1 dBm. The chosen slot time is 60 ms.

We compared the results regarding RSSI values and energy consumption between our "golden model" and the combined framework (MiXiM, PAWiS, and STEAM-Sim). The "golden model" is build upon STEAM-Sim which integrates PAWiS and was verified by means of measurements in [2]. To ensure comparable results we had to include the PAWiS free space propagation model as described in [4] in the combined framework. The new analogue model was configured with attenuation exponent b equal to two and the effective antenna area equal to 9.87670 cm^2.

A. Static Scenario

In this scenario the base station was placed in the center of a 100x100 m^2 area. The nine sensor nodes were placed arbitrarily in this area. The results for the RSSI values and the consumed energy of the "golden model" and the combined simulation framework matched exactly.

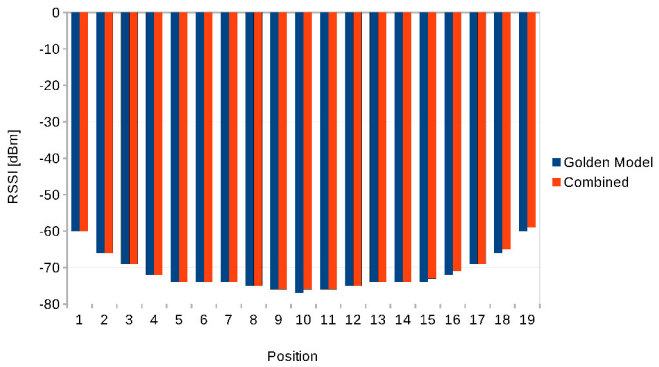
B. Mobile Scenario

The second evaluation scenario incorporates a mobile sensor node and therefore requires the usage of a mobility pattern. We used the rectangular movement pattern provided by MiXiM, i.e., a mobile sensor node moves with ten metres per second anti-clockwise through nineteen positions describing a rectangle. At every position it sends a packet to a fixed positioned base station. PAWiS does not provide mobility patterns, so we implemented such a movement by means of LUA scripts in the "golden model". Since we do not use a linear continuous movement (i.e., we set the positions discrete) the simulation results differ from each other as can be seen for position 10 in Fig. 4(a). Further, the MiXiM rectangular movement pattern introduces a random deviation from the starting position. Nevertheless, the absolute error is 1 dBm which corresponds to the resolution of the modeled hardware.

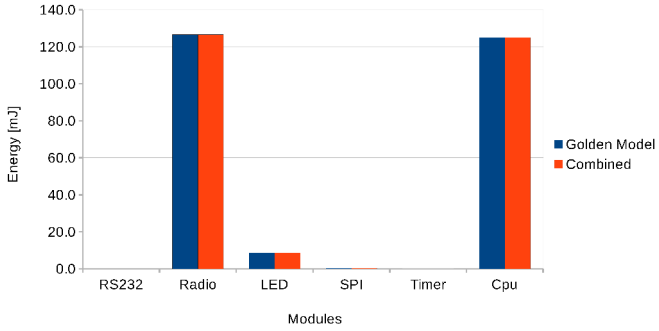
We further simulated the energy consumption of the mobile sensor node as depicted in Fig. 4(b). As can be seen, the results between the "golden model" and the combined framework match exactly.

IV. Conclusions

In this paper we identified shortcomings in state-of-the-art simulation environments of WSNs, namely focusing on special characteristics of WSNs such as mobility and realistic channels



(a) RSSI readout values



(b) Consumed energy of the mobile sensor node

Fig. 4. Evaluation results for the integrated framework

models. We have presented a way towards an integration of well-known, established, and daily used simulation frameworks. MiXiM especially offers mobility and sophisticated channel models whereas PAWiS provides accurate modeling of the timing and energy consumption of hardware components. The STEAM-Sim simulator enables the time accurate simulation of real-life application code in an efficient way. Our system integration approach, which was verified by two simulation studies, will give the community the opportunity to establish accurate simulation of mobile WSNs and WSANs.

V. Acknowledgements

This work has been supported by the Linz Center of Mechatronics in the framework of the Austrian COMET-K2 programme.

References

[1] A. Köpke, M. Swigulski, K. Wessel, and et. al. Simulating Wireless and Mobile Networks in OMNeT++ – the MiXiM Vision. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques*, SIMUtools'08, 2008.

[2] G. Möstl, R. Hagelauer, G. Müller, and A. Springer. STEAM-Sim: Filling the Gap Between Time Accuracy and Scalability in Simulation of Wireless Sensor Networks. In *Proceedings of the 8th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, PM2HW2N'13, 2013.

[3] A. Varga. The OMNeT++ Discrete Event Simulation System. *Proceedings of the European Simulation Multiconference (ESM)*, 2001.

[4] D. Weber, J. Glaser, and S. Mahlknecht. Discrete Event Simulation Framework for Power Aware Wireless Sensor Networks. In *5th IEEE International Conference on Industrial Informatics*, 2007.

OMNeT++ and mosaik: Enabling Simulation of Smart Grid Communications

Position Paper

Jens Dede*, Koojana Kuladinithi*, Anna Förster*, Okko Nannen†, Sebastian Lehnhoff†
*Sustainable Communication Networks Group, University of Bremen, Germany
Email: {jd, koo, afoerster}@comnets.uni-bremen.de
†R&D Division Energy, OFFIS, Germany
Email: {okko.nannen, sebastian.lehnhoff}@offis.de

***Abstract*—This paper presents a preliminary system architecture of integrating OMNeT++ into the mosaik co-simulation framework. This will enable realistic simulation of communication network protocols and services for smart grid scenarios and on the other side, further development of communication protocols for smart grid applications. Thus, by integrating OMNeT++ and mosaik, both communities will be able to leverage each others's sophisticated simulation models and expertise.**

The main challenges identified are the external management of the OMNeT++ simulation kernel and performance issues when federating various simulators, including OMNeT++ into the mosaik framework. The purpose of this paper is to bring these challenges up and to gather relevant experience and expertise from the OMNeT++ community. We especially encourage collaboration among all OMNeT++ developers and users.

I. Introduction

Smart grids have been identified as the most important development towards renewable energy production and usage. Still in their baby shoes, they exhibit a number of challenges in a very wide spectrum, reaching from market strategies through new power grid networks to communication.

Mosaik [1] is a flexible smart grid co-simulation framework. It allows to combine several simulators for different requirements. Using mosaik, it is possible to combine simulators for photovoltaic, power plants, households etc. to a complex smart grid simulation scenario. However, mosaik assumes a perfect link for the communication between the individual entities like houses, power plants and electric vehicles.

OMNeT++ focusses on the communication link and offers a huge variety of toolboxes for the majority of common communication technologies.

The purpose of this position paper shows a possible way of connecting OMNeT++ and mosaik to get an easy-to-use framework for a realistic simulation of the communication and the power flows of current and future smart grid scenarios.

II. Related works

Several publications are available which introduce co-simulation solutions for the power grid and communication links. Most of the work highlight the integration of the different simulator types as the main issue: Most power grid simulators operate using discrete time simulation whereas communication network simulators discretise on an event base (discrete event simulation). One task for co-simulation frameworks is to combine both simulator types. Without claiming completeness, we point out some exemplary existing solutions.

The authors in [2] simulate the influence of cyber-attacks on a Smart Grid using Matlab/Simulink in combination with OPNET. Due to the change of the license policy of riverbed (manufacturer of OPNET) for academic users, OPNET has lost however its attractiveness for universities. In [3], a generic co-simulation framework called "*FNCS*" is introduced. It can connect power grid and communication network simulators. The evaluation of the framework is performed using GridLAB-D (distribution simulator), PowerFlow (power flow simulator) and ns-3 (communication simulator). It can be extended to use other simulators. Additional time synchronisation algorithms for FNCS were introduced in [4]. Our own extensive work on communication protocols focuses on OMNeT++ as a tool and thus transferring is not feasible. Anyway, some possible solutions to overcome co-simulation issues regarding discrete time and discrete event simulation are discussed in FNCS publications and might be helpful for the desired extension of OMNeT++.

Also for OMNeT++, co-simulation frameworks for the smart grid are available. In [5], OMNeT++ and OpenDSS (electric power Distribution System Simulator) are combined and OMNeT++ takes control of OpenDSS. In [6], OMNeT++ and OpenDSS are run in parallel and the events are synchronized at certain time slots. Both solutions are limited to OpenDSS for the grid simulation and not easily extendible for additional simulators.

In [7], the authors use OMNeT++ to analyze measurements from a real testbed to evaluate the communication effort caused by using electric vehicles for stabilizing the power grid. This shows a good example of the flexibility of OMNeT++ regarding combination with other information sources.

In summary, the existing co-simulation frameworks support only few power grid simulators like OpenDSS, GridLAB-D and PowerFlow. Mosaik offers a framework with support for a considerable number of simulators. Plus, it offers simulation servers and support. Combining OMNeT++ and mosaik will result in a flexible and powerful co-simulation framework not only for the power aspects but also for the required communication networks of future power grids. Furthermore, this

integration will utilise a variety of communication technologies and protocols already available on OMNeT++ to be evaluated for future applications in the area of power grids.

III. Overview of mosaik

Smart grids comprise a vast number of active components and participants that have to fulfil the task of keeping power demand and supply in balance while adhering to delicate operational constraints, which are required to maintain a safe and a stable power supply. Further, the domain space of future energy systems, so called "Smart Grids" are closely interlinked with other complex systems as given below.

- environmental weather conditions (influencing wind and solar power feed-in)
- energy markets (influencing unit commitment and large-scale generation schedules)
- socio-technical system (influencing end-user-operated appliances as well as small scale generation units)
- information and communication systems (influencing the availability and accuracy of operational telemetry data necessary for a stable and efficient operation)

Sophisticated models and simulators exist for each of the above mentioned systems that have been researched and expensively developed by respective domain experts. Though some of the existing smart grid simulation tools have extended their scope to consider other relevant systems (e.g. power system plus communication system simulation), these setups are usually limited in functionality of the whole spectrum of future energy domain. To the best of our knowledge, no simulators yet exist that are capable of producing the entire domain spectrum for smart grid applications [8]. This is simply because of complexity in integrating above mentioned systems developed individually to model more specific functions. However, several existing co-simulation suites exist as mentioned in the related work.

The SESA-Lab at the University of Oldenburg aims at bridging this gap by providing a collaborative simulation and integration platform that consists of sophisticated adapters/interfaces to many interdisciplinary tools and simulators in the domain space of future energy systems. It is continuously extended to functionally interlink domain-specific models that have not yet been able or used to run in an integrated fashion with other sophisticated simulators. The proposed simulation framework is called mosaik that not only provides interfaces to the above mentioned systems, but a scheduling mechanism that is capable of efficiently orchestrating the communication and information exchange between heterogeneous models and simulators (hard- and software) interoperated in this fashion. Mosaik allows you to reuse and combine existing simulation models and simulators to create large-scale Smart Grid scenarios – i.e. thousands of simulated entities distributed over multiple simulator processes. These scenarios can then serve as a test bed for various types of control strategies (e.g., multi-agent systems (MAS) or centralized control). This is shown in Figure 1 which presents the overall system architecture of mosaik with several connected simulators.

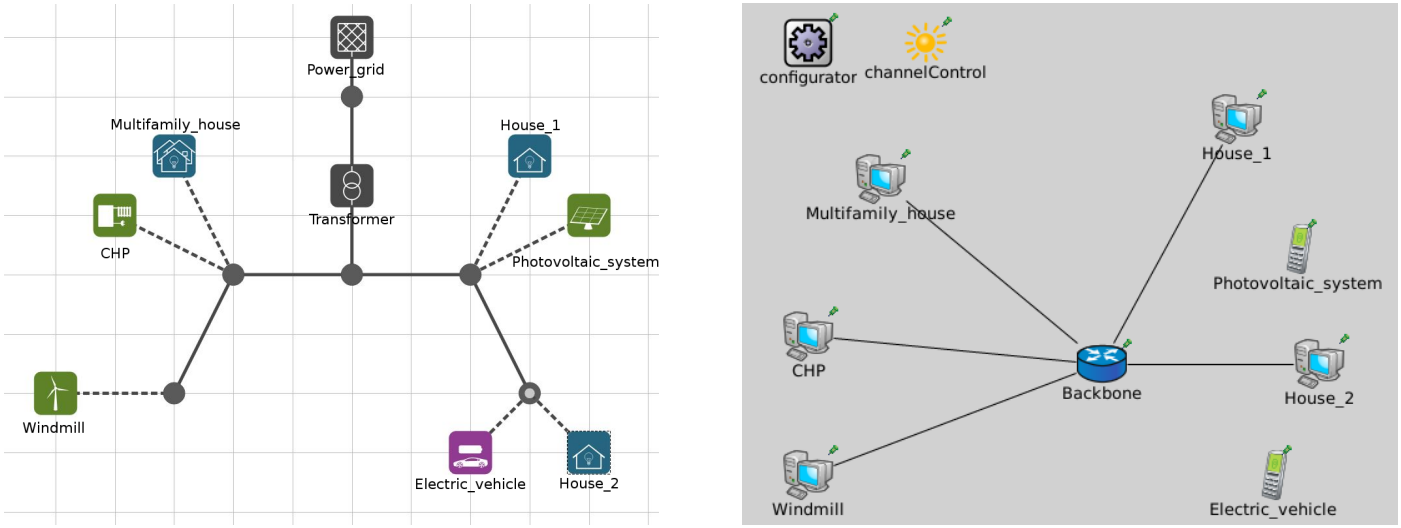
Fig. 1. The mosaik system architecture: mosaik orchestrates all connected simulators

Mosaik assumes a perfect communication link between the separate simulated entities. To overcome this limitation, we will extend OMNeT++ with mosaik interfaces to simulate realistic links. Mosaik is written in Python and completely open source (under LGPL v2.0), including some simple simulators, a binding to PYPOWER and a demonstration scenario. More information can be found at [1].

IV. Overview of OMNeT++

This paper focuses on how to enable the integration of OMNeT++ into the mosaik environment. We will not go into details about the OMNeT++ simulation environment here, since we assume a high degree of expertise in the audience. OMNeT++ is a discrete event simulator for simulating communication/interaction between some network entities or agents. Typically these agents are fully controlled by OMNeT++, even if their implementation might be (partially) external, e.g. through additional libraries. The main reason of doing this is the nature of discrete event simulators, where events are inserted from all communicating agents into one single event queue, which is processed then sorted in terms of simulation time.

Currently, there are possibilities to embed OMNeT++ simulations into other applications, which means in particular that the simulation can be started externally and results can be fed into the calling application automatically. Furthermore, as already mentioned above, individual agents can be implemented externally, but are controlled from inside the OMNeT++ simulation. For example, one can use an implementation of a battery behaviour from an external library, which is called from inside a corresponding OMNeT++ module. This makes it very easy to integrate simulations **into** OMNet++, but not the other way around. There are no possibilities to control OMNeT++ externally, e.g. calling it event by event or inserting events from other applications. At the same time, the highly modularised architecture of OMNeT++ makes this more an implementation question rather than a technical limitation or real challenge.



(a) The mosaik representation

(b) The OMNeT++ representation

Fig. 2. Two representations of one scenario containing a windmill-powered plant, a combined heat and power unit (CHP), one multifamily house, a transformer, an uplink to the upper layer power grid as a reference, two houses, one photovoltaic system and an electric vehicle. (a) depicts the corresponding mosaik representation while (b) shows the same in OMNeT++. Note: The transformer and the upper layer power grid are not depicted as they are assumed to not have communication capabilities.

V. Preliminary System Architecture

The system architecture of mosaik is depicted in Figure 1. The communication between simulators is enabled in a direct way through the simulation kernel itself. This means, if two different entities in a simulation need to exchange information, they do so by simply sending the information to the other entity through mosaik. No delays, protocol details, etc. are considered. The corresponding API was published first in [9].

In order to adhere to the general system architecture of mosaik, OMNeT++ needs to be connected as a mosaik component through implementing the mosaik API. Then, communication between entities will be forwarded to OMNeT++ for simulation instead of directly forwarding to the receiver entity. Besides the implementation and integration of the mosaik API, the time synchronization of mosaik and OMNeT++ has to be considered. While OMNeT++ performs a discrete event simulation, mosaik performs a discrete time simulation. The synchronization of both types of simulation is for example discussed in [3] and [4]. Besides the time synchronization, several additional adaptations are required:

A. Extracting the relevant communication network from mosaik to OMNeT++

Figure 2 depicts an example scenario consisting of a windmill-powered plant, a combined heat and power unit (CHP), one multifamily house, a transformer, an uplink to the upper layer power grid as a reference, two houses, one photovoltaic system and an electric vehicle. The power electric connections of all entities are show in the topology representation of mosaik as depicted in Figure 2(a). Mosaik does not consider the communication links. Figure 2(b) shows a possible representation of the communication links using OMNeT++ where we assume that the transformer and the upper layer power grid have no communication capabilities and therefore are not shown. Furthermore, the electric vehicle and the photovoltaic system are assumed to communicate wirelessly as entities like the photovoltaic system might be placed in a remote place without wired connectivity. An electric vehicle may require a wireless connection to request load capacities, i.e. while driving home.

To co-simulate the communication using mosaik, the communication capabilities of all entities in mosaik are required and need to be transfered to OMNeT++. We consider this step is only moderately challenging, as it only includes extensions to mosaik, which fully conform to its current design. From the side of OMNeT++, a NED description can be extracted relatively easy from the mosaik scenario description and thus easily forwarded to OMNeT++.

B. Full external control of OMNeT++ simulation

In order to give the full control over the simulation to mosaik to orchestrate all simulators at the same time, OMNeT++ needs to be controlled externally. Currently, there is no obvious way to do this. The only relevant feature is the possibility to embed the simulation, but this does not give the control to an external party. Rather, we need to do the following:

- First initialise the simulation
- Run events one by one and pass relevant results to mosaik
- Insert new events externally from mosaik (e.g. a new data packet or stream)
- Forward errors to mosaik
- Finish the simulation

As mentioned above, the exchange of events between mosaik and OMNeT++ requires a synchronization between the mosaik time base and the OMNeT++ event base. We plan to implement this by extending the OMNeT++ kernel with

methods similar to the already existing `simulate()` function, such as `simulate_with_control()`, `step_one_event()` and `simulate_until()`.

Besides the full external control of OMNeT++, a real-time simulation of both simulators might be feasible. In this case, mosaik could insert events at the application layer to the OMNeT++ simulator. But this may lead to the following issues:

- As the runtime of targeted smart grid simulations can be long (i.e. in the order of month or even years), the simulation will be long as well.
- For complex simulation models and scenarios, the real-time simulation might be not applicable due to the limitation of processing power (simulation time longer than real time).

C. Improve performance of federated simulations

The above described idea of running OMNeT++ externally from mosaik event by event is not very efficient. It will require mosaik to call OMNeT++ quite often, even if no relevant events are executed. For example, if two entities are trying to communicate through OMNeT++ using a UDP connection, the only relevant events for the entities are the passing of the data to the connection and the received data on the other side. All protocol-level events, such as queueing, medium access, etc. do not have to be exposed to mosaik. Therefore, several options can be used to optimise the performance:

- Pass results to mosaik only at the OMNeT++ application level, e.g. as defined in the INET framework. All other events are kept internal and are not communicated to mosaik. This results in the issues mentioned in Section V-B.
- Enable multiple event simulation in OMNeT++. The main challenge lies in the fact that OMNeT++ does not know when the next communication is due, as this is in control of mosaik. Thus, mosaik needs to resolve in its kernel when the next communication is due and to allow OMNeT++ to run so far.
- Identify independent communications. Mosaik needs to explore its scenario and to identify communication islands. In this case, several smaller OMNeT++ simulations can be started and executed in parallel. But it is a challenging task to explore these communication island in a reliable way.
- Make use of existing parallelization techniques for OMNeT++. This will speed up OMNeT++ itself and thus also the general mosaik scenario.

VI. Conclusion

In this position paper, the integration of OMNeT++ into the mosaik co-simulation framework was described. Several pitfalls and required adaptations were brought up. However, no serious limitations or expected problems have been identified and we believe that the federation of the testbeds is feasible. The main issue thus remains the performance of the federated simulations.

References

[1] "Mosaik is a flexible smart grid co-simulation framework." accessed: 2015-06-23. [Online]. Available: http://mosaik.offis.de

[2] M. A. H. Sadi, M. H. Ali, D. Dasgupta, and R. K. Abercrombie, "Opnet/simulink based testbed for disturbance detection in the smart grid," in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, ser. CISR '15. New York, NY, USA: ACM, 2015, pp. 17:1–17:4. [Online]. Available: http://doi.acm.org/10.1145/2746266.2746283

[3] S. Ciraci, J. Daily, J. Fuller, A. Fisher, L. Marinovici, and K. Agarwal, "Fncs: A framework for power system and communication networks co-simulation," in *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, ser. DEVS '14. San Diego, CA, USA: Society for Computer Simulation International, 2014, pp. 36:1–36:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665008.2665044

[4] S. Ciraci, J. Daily, K. Agarwal, J. Fuller, L. Marinovici, and A. Fisher, "Synchronization algorithms for co-simulation of power grid and communication networks," in *Modelling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on*, Sept 2014, pp. 355–364.

[5] M. Lévesque, D. Q. Xu, G. Joós, and M. Maier, "Communications and power distribution network co-simulation for multidisciplinary smart grid experimentations," in *Proceedings of the 45th Annual Simulation Symposium*, ser. ANSS '12. San Diego, CA, USA: Society for Computer Simulation International, 2012, pp. 2:1–2:7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2331751.2331753

[6] D. Bhor, K. Angappan, and K. Sivalingam, "A co-simulation framework for smart grid wide-area monitoring networks," in *Communication Systems and Networks (COMSNETS), 2014 Sixth International Conference on*, Jan 2014, pp. 1–8.

[7] S. Bocker, C. Lewandowski, C. Wietfeld, T. Schluter, and C. Rehtanz, "Ict based performance evaluation of control reserve provision using electric vehicles," in *Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), 2014 IEEE PES*, Oct 2014, pp. 1–6.

[8] M. Buscher, A. Claassen, M. Kube, S. Lehnhoff, K. Piech, S. Rohjans, S. Scherfke, C. Steinbrink, J. Velasquez, F. Tempez, and Y. Bouzid, "Integrated smart grid simulations for generic automation architectures with rt-lab and mosaik," in *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, Nov 2014, pp. 194–199.

[9] S. Schütte, S. Scherfke, and M. Sonnenschein, "mosaik-smart grid simulation api," *Proceedings of SMARTGREENS*, pp. 14–24, 2012.

Federating OMNeT++ Simulations with Testbed Environments

Asanga Udugama*, Koojana Kuladinithi†, Anna Förster*, Carmelita Görg*
*Sustainable Communication Networks Group, University of Bremen, Germany
Email: {adu|anna.foerster|cg}@comnets.uni-bremen.de
†Institute of Communication Networks, Hamburg University of Technology, Germany
Email: {koojana.kuladinithi}@tuhh.de

***Abstract*—We are in the process of developing a system architecture for opportunistic and information centric communications. This architecture (called Keetchi), meant for the Internet of Things (IoT) is focussed on enabling applications to perform distributed and decentralised communications among smart devices. To realise and evaluate this architecture, we follow a 3-step approach. Our first approach of evaluation is the development of a testbed with smart devices (mainly smart phones and tablets) deployed with this architecture including the applications. The second step is where the architecture is evaluated in large scale scenarios with the OMNeT++ simulation environment. The third step is where the OMNeT++ simulation environment is fed with traces of data collected from experiments done using the testbed. In realising these environments, we develop the functionality of this architecture as a common code base that is able to operate in the OMNeT++ environment as well as in the smart devices of the testbed (e.g., Android, iOS, Contiki, etc.). This paper presents the details of the "Write once, compile anywhere" (WOCA) code base architecture of Keetchi.**

I. Introduction

The use of networks is moving in the direction of being information centric. International Data Corporation (IDC) predicts, that by 2020, the digital universe will have close to 45 ZB of data [1]. In the same report of current and future trends, IDC predicts further that by 2020, the total number of connectable devices in the Internet of Things (IoT) will be above 200 billion. To address these trends and the different application areas, new communication architectures and protocols have to be designed. One such architecture that we are building is Keetchi. It addresses the opportunistic communications paradigm with the focus on distributed, decentralised and reinforcement learning based [2] [3] communications.

Keetchi is meant to operate in a number of different environments focusing on large scale networks. Firstly, in infrastructure-less environments, nodes deployed with the Keetchi architecture will perform direct communications with other nodes. Secondly, when infrastructure is present, nodes may use the infrastructure to reach other nodes. Thirdly, Keetchi based nodes may also operate in hybrid environments where infrastructure-less environments interact with infrastructure based environments. When developing this architecture and its functionality, our intensions go beyond designing and evaluation of efficient mechanisms. On the one hand, we wish to develop code that is platform independent and is able to be executed not only in OMNeT++, but also on real IoT based devices. Currently, we are building a testbed consisting of hundreds of IoT based devices installed with Android, iOS, Contiki and Embedded Linux operating systems. On the other hand, we are also interested in large scale evaluations, which can only be realised with simulations. At the same time, simulations need to be sophisticated with traces from real applications and environments to make the results more realistic and representable.

Therefore, in this paper we present our on-going work on developing a platform independent code base of Keetchi that enables distributed, decentralised and reinforcement learning based communications in OMNeT++ as well as in the smart devices deployed in a testbed. As the focus of this paper is the code base, we do not describe the Keetchi architecture in detail. But, a brief description of the salient aspects of the Keetchi architecture is provided in the next section to assist the understanding of the subsequent sections.

II. System Architecture of Keetchi

At the heart of Keetchi is the emphasis on exploiting the benefits of information centricity of today's communications. It has the following main features:

- **Information Centric Communications** - Consideration of named data as in Content Centric Networking (CCN)[4] and placing less emphasis on naming hosts (current networks),
- **Distributed Control** - Every device is in charge of its own operations without any dependance on centralised controls,
- **Distributed Caching** - Devices capable of storing data select *when* and *what* to store based on local decisions,
- **Reinforcement Leaning Based Data Handling** - Feedback for exchanged data decides what data is considered for propagation in the network and what data is stored in caches,
- **Opportunistic Data Propagation** - Constant monitoring of the neighbourhood (i.e., to build a connectivity map) is performed to identify opportunities for communications and infrastructure-less communications.

The Keetchi architecture which is deployed in mobile devices (i.e., Keetchi nodes) and used by the applications in those mobile devices, consists of a number of modules that interact to store and propagate data generated by the applications.

Figure 1 shows an example of the message propagations in the Keetchi architecture in the time domain. *Node A* propagates data for which feedback messages are received (*Node B* and *Node C*). Once *Node B* meets *Node E*, it can re-propagate the

data as *Node E* had previously indicated a preference for such data. The scenario shown in Figure 1 is for an infrastructure-less deployment which could employ bearer technologies such as Wifi Direct or Bluetooth Low Energy (BLE) to communicate in a peer-to-peer communication model. A deployment in an infrastructure based environment of Keetchi may use WiFi or a 3rd Generation Partnership Program (3GPP) technology.

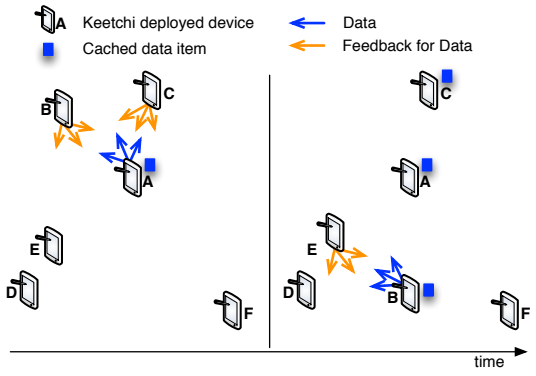


Fig. 1. Operations in a Keetchi based network

Every Keetchi node has a 3-layer protocol stack. These layers perform the following activities.

- **Application Layer** - The application layer provides the support required to host applications that are based on decentralised and distributed direct communications.
- **Keetchi Layer** - The Keetchi layer handles the functions of maintaining the different data dissemination strategies together with the communication models employed and data caching based on the features detailed above.
- **Link Layer** - The Keetchi architecture leverages link technologies such as Wi-Fi Direct or BLE to perform direct communications.

Communication in the Keetchi architecture is performed using 2 message types. They are:

- **Data Message** - The Data message[1] carries the named data generated by applications executed in nodes using the Keetchi architecture. These messages are propagated through the network opportunistically and controlled by the reinforcement learning caching and forwarding model.
- **Feedback Message** - The Feedback message[1] carries information that evaluates previously received named data or as a means of soliciting additional data. Its serves as a reward to the reinforcement learning model.

The work presented in [5] describes a model and the evaluation of a peer-to-peer content-centric model for multi-group communications with WiFi Direct. [6] presents a study on feasibility, advantages and challenges of an IoT based on Named Data Networking (NDN). The Keetchi architecture includes some of the aspects considered in [5] and [6] (i.e., information centric communications, IoT, WiFi Direct, etc.). But the differentiating aspect of Keetchi with the aforementioned work is its additional focus on opportunistic communications and reinforcement learning based data handling which are vital for communications in the IoT.

[1]The term *Data* message has the same semantic meaning as in CCN with the difference being (due to distributed infrastructure-less communications) that Data messages are sent out opportunistically without receiving a CCN *Interest* message for this data first. Due to these differences the second type of message used in Keetchi is called the *Feedback* message which reverses the way interests for data are announced: it evaluates already received data in order to either reinforce its dissemination or inhibit it. This forms the basis of the Keetchi reinforcement learning algorithm.

III. Code Base Architecture

The Keetchi architecture is envisioned to be deployed in a number of different computing platforms. They range from ubiquitous devices such as smart phones to custom built computing devices for rugged environments. Additionally, the Keetchi architecture is envisaged to be evaluated for its performance in large scale simulations in OMNeT++.

To cater to these 2 requirements, i.e., deployment in testbed environments and evaluation in OMNeT++, the functionality related to performing the operations unique to the Keetchi layer are developed as a common codebase. Figure 2 shows the architecture of the codebase with some example platforms on which it is expected to be executed.

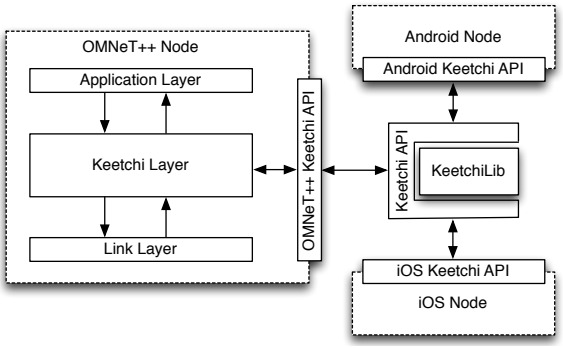


Fig. 2. Codebase Architecture

The functionality of the Keetchi layer operations are placed in a library, *KeetchiLib*. Each computing platform has an API that calls the functionality of Keetchi operations. The KeetchiLib is developed using C++ and the APIs handle the translations between C++ and the native languages used in those computing platforms. For example, in the Android platform, the API converts between Java and the C++ code using the Java Native Interface (JNI) functionality. Depending on the architecture of how applications are enabled in a specific computing platform, the KeetchiLib is used as a dynamic or a static library.

With OMNeT++, the API is a simple pass through API and the KeetchiLib could be configured to be used either as a dynamic or a static linked library.

The KeetchiLib consists of a number of classes that are instantiated and released during operations. The main class, KeetchiMain, is instantiated at and held during the lifetime of each single Keetchi node. The Keetchi Layer of each of the computing platforms must instantiate the KeetchiMain class to access all Keetchi functionality. This functionality consists of code to handle data store operations and a set of algorithms including other functions (described later). The data stores are implemented as hash maps to store the data that traverse the Keetchi node and a map of neighbours (all nodes met

```
/**
 * Process an incoming Feedback message from either the application
 * layer or the link layer.
 *
 * @param fromWhere   The message source, application layer or link
 *                    layer.
 * @param feedbackMsg The Feedback message with its contents as a
 *                    KLFeedbackMsg instance.
 * @param currentTime The current clock time.
 * @return            A KLAction instance with all actions required
 *                    to be taken by the Keetchi layer.
 */
KLAction* processFeedbackMsg(int fromWhere, KLFeedbackMsg *feedbackMsg,
                                                    double currentTime);
```

Fig. 3. An example method of the KeetchiMain class

up to now). The algorithms in the KeetchiLib are for cache management, for execution of the dissemination strategies and for activating the different communication models.

Figure 3 shows a method definition of one of the functions of the KeetchiMain class. This method processes an incoming Feedback message from either the application layer or the link layer (*fromWhere*). Once processed, the method returns a KLAction type instance with instructions on what actions must be taken by the Keetchi layer and the information required for this. These actions may range from taking no action to sending multiple messages to the application and link layer.

An important aspect in the KeetchiLib is the use of the current clock time. The clock time is required in multiple activities of Keetchi such as for time stamping messages. Since simulators operate on the simulation time, the system clock is never used in KeetchiLib. Instead, the current time is passed in all methods (where it is required) as a parameter.

The functionality exposed in the KeetchiLib for OMNeT++ fall mainly into 4 categories. These are as follows:

- **Incoming Message Processing** - As described above, operations at Keetchi nodes result in the arrival of Feedback and Data messages at other Keetchi nodes. These messages trigger the Keetchi layer to perform other actions.
- **Opportunistic Message Generations** - In addition to the above messages (Feedback and Data messages), another trigger for activity at the Keetchi layer is the receipt of neighbourhood node information. These will trigger the generation of Data messages.
- **Status Information Servicing** - KeetchiLib creates and maintains a number of data stores in its operations. This set of functions provide access to the information held by these data stores.
- **Wiring for Statistics** - The functions related to dumping statistical information is coupled with the functions of the previous function categories. The return action information of the previous functions also bring back statistical information that could be used to generate statistics in OMNeT++.

IV. Keetchi OMNeT++ Model

A Keetchi node in OMNeT++ implements the 3-layer protocol stack (shown in Figure 2) together with the use of mobility models, traffic models (application events) and wireless propagation models (physical layer) available in OMNeT++. Figure 4 shows the implementation of the protocol stack and the connections between the protocol layers of a Keetchi node in OMNeT++. There are a number of applications considered

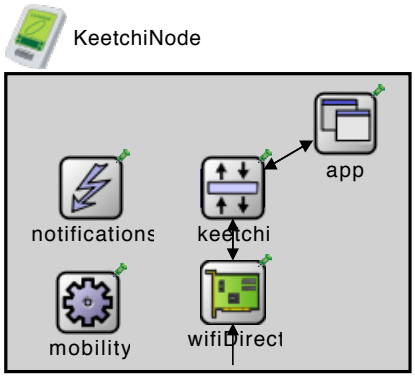


Fig. 4. OMNeT++ Keetchi Node Architecture

for the Keetchi environment. They range from emergency messaging applications to social networking. There are 3 types of applications in general. *Traffic Generators* produce data that expect feedback to make decisions on the type of data to generate and the recipients to send to, subsequently. *Traffic Consumers* are the users of data who generate feedback based on preferences. The third type of applications are the hybrid applications that perform both of the above (send and receive).

When considering sources of traffic (application events), we consider applications that generate traffic based on the synthetic models available in OMNeT++ or, based on the traces obtained from our IoT testbed.

The Keetchi layer of a node is responsible for providing the Keetchi functionality through the KeetchiLib. Using the *OMNeT++ Keetchi API*, the Keetchi layer creates an instance of a KeetchiMain class during the intitialisations (*initialize()*) using a multi-stage initialisation process. The Keetchi layer will receive Feedback and Data messages from the applications and the Link layer which are passed on to the OMNeT++ Keetchi API for processing. The C++ class in OMNeT++ for the Keetchi layer (*Keetchi.cc*) converts the properties in the messages into the format of the message required by the KeetchiLib. Once processed, KeetchiLib will return the actions to be taken by the OMNeT++ Keetchi layer. These actions may result in generating multiple messages to be sent to either the applications, the Link layer or both.

Another type of information passed on to the KeetchiLib is the information on nodes currently reachable in the neighbourhood. This information, which is received from the OMNeT++ Link layer models is used by the KeetchiLib to build a map of the current nodes in the vicinity and to generate Data messages which have been previously indicated as being of interest (opportunistic communications).

The Link layer in the Keetchi architecture is realised using a Wi-Fi Direct bearer. The WiFi Adhoc implementation of the INET framework of OMNeT++ is being extended to include the functionality related to Wi-Fi Direct operations. In addition to providing functionality to establish connections with multiple other Keetchi nodes, the Wi-Fi Direct extensions also include the provisioning of information of the nodes currently in the neighbourhood. As described above, the Keetchi layer is overlaid over a Wi-Fi Direct Link layer. Therefore, the Keetchi layer uses the Media Access Control (MAC) addresses

of nodes to identify a node uniquely.

V. A Simulated Scenario and Evaluation Metrics

The Keetchi architecture is envisioned to be deployed in a number of application areas. These areas relate to emergency services, social networking, smart building controls, etc. One such application is an exchange service for "hand-me-down" goods. This application, called *UniRecycler* allows users to inform about goods that are available to give away and for users who are interested in these goods to inform back of their interest and feedback.

The UniRecycler application uses the peer-to-peer, decentralised and distributed communication model of the Keetchi architecture. Additionally, it uses data caching and opportunistic communications of the architecture to perform store-and-forward communications. When evaluating such an application in OMNeT++, we consider large scale network scenarios with many individual devices. The nodes in this scenario would connect to other nodes in their vicinity, perform communications, and disconnect again due to mobility or other issues. The nodes that move away may act as carriers of data collected on behalf of other nodes, to be transferred if they connect later.

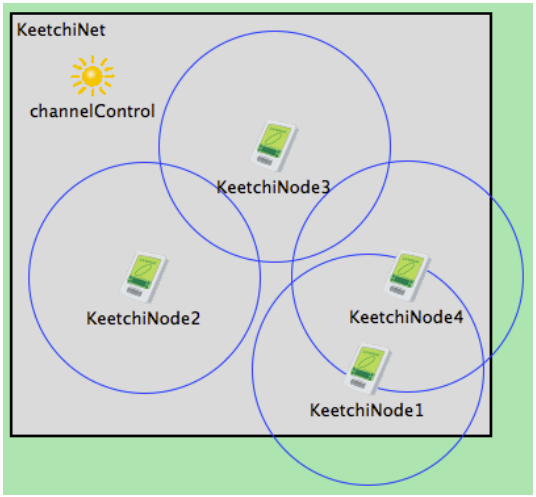


Fig. 5. An OMNeT++ Scenario with Keetchi Nodes

Figure 5 shows the operation of a small scenario with multiple nodes deployed with the Keetchi architecture. The user of *KeetchiNode1* may inform of a give-away through a Data message which gets propagated to the *KeetchiNode2* over the Wi-Fi Direct based connections of *KeetchiNode4* and *KeetchiNode3*. When the Data message is received, some of these nodes (e.g., *KeetchiNode3*) may send a Feedback message indicating its interest in such data. Further, if *KeetchiNode2* decides to disconnect and connect to other clusters, it may decide to resend the original Data message from its cache based on previously received feedback for such data.

When considering large scale scenarios, evaluations in testbed environments are insufficient due to the limitations of the scale related issues of testbeds (e.g., number of devices, connection ranges, etc). Moreover, even in small scale networks that could be deployed with testbeds, limitations of the used devices may hamper proper evaluations (e.g., availability of only one type of bearer technology).

On the other hand, OMNeT++ based simulated scenarios provide the flexibility to create networks with extremely large node numbers spanning large geographical areas and the ability to have full-featured devices for evaluations. Evaluating the Keetchi architecture in such scenarios is required to understand the behaviour of mechanisms used in all parts of the Keetchi architecture. Therefore, we consider a number of evaluation metrics that focus on the following areas:

Feedback System: Feedback messages carry a numeric weight that is computed at each node. This metric should consider the popularity, freshness, location and the usefulness of data. Algorithms such as Q-learning are employed by the Keetchi architecture to compute these weights. Large scale evaluations provide the possibility of understanding the influences on the weights depending on the node density and mobility.
Opportunistic Data Propagation Strategies: One of the novelties of the Keetchi architecture is its self adaptiveness employed in propagating data depending on the available communication opportunities and feedback from neighbours. The ability of OMNeT++ based scenarios to consider full-featured devices (e.g., devices with Wi-Fi Direct and BLE together) in simulations has the possibility of evaluating the performance of opportunistic communication over multiple bearers simultaneously.
Caching: Distributed caching, cache policies, cache sizes and the activities of nodes influence what data is ultimately held in caches. The OMNeT++ based scenarios considering scale networks and full-featured devices (e.g., larger caching capabilities) with varying traffic patters help in understanding the issues of scalability in Keetchi architecture based networks.
Mobility Patterns: There are many mobility patterns to consider when evaluating the Keetchi architecture. These patterns depend not only on the application type but also on many other factors (e.g., contact time, data prioritisation, etc.) Due to the flexibility of configuration, the influence on performance of these mobility patterns are best evaluated in a simulated environment compared to testbed environments.

VI. Technical Challenges

The Keetchi layer which implements the Keetchi architecture is impervious of the underlying networking bearer technologies. The underlying link layer technology used is, in general, required to perform hop-by-hop, peer-to-peer communications and provide information of the neighborhood. In our work, we focus on WiFi Direct and BLE technologies. There are a number of challanges that have to be addressed to use these technologies in the Keetchi architecture. In the following is a brief description of these challenges.

WiFi Direct - WiFi Direct is a standard of the WiFi Alliance to perform peer-to-peer (or multi-peer) communications. It is an extension to the functionality of WiFi. WiFi Direct requires the establishment of a trusted connection between peers before any communications could occur. Current implementations of WiFi Direct (mainly in the Android platform) require re-exchanging of trust credentials through user interactions when re-establishing communication sessions. When exchanging Keetchi messages (using WiFi Direct), we consider the re-exchanging of trust credentials to be a drawback .

Bluetooth Low Energy - Bluetooth Low Energy (BLE) provides a very energy efficient means of performing peer-

to-peer communications. When compared to the operations of Bluetooth, the available bandwidth of BLE is lower. BLE is especially meant for small message exchanges. But, when considering the communication patterns of Keetchi, there is a requirement to transmit messages with large payloads (e.g., images). Therefore, the link layer used by the Keetchi architecture must be able to select the bearer based on the type of message to send (i.e., cross-layer control).

VII. Conclusion

Services envisaged for the IoT require the development of new communication architectures. Keetchi is a communication architecture meant for IoT based ubiquitous devices to host some of these services. The Keetchi architecture, which exploits the benefits of opportunistic communications and the information centricity of today's network use, provides the means for applications to perform peer-to-peer, distributed and decentralised communications. When developing and evaluating such an architecture, our focus has been to develop a code base of the functionality that is able to be used not only on the implementations for the different operating systems but also in OMNeT++. We are in the process of building such a code base and this paper presents this on-going work in the context of OMNeT++. By presenting this work, firstly, we wish to share our experiences and secondly, to discuss with the community of OMNeT++ users and developers about the features that we intend to build (e.g., mobility models for opportunistic communications, WiFi Direct and BLE) which are mutually beneficial for our work as well as for the community.

References

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

[2] A. Förster, K. Kuladinithi, A. Timm-Giel, C. Görg and S.Giordano, DICE: A Novel Platform to Support Massively Distributed Clouds, in Proceedings of the International Conference on Mobile Networks and Management (MONAMI), Cork, Ireland, 2013.

[3] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[4] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs and R. Braynard, *Networking Named Content*, in proceedings of the CoNEXT - 5th International conference on Emerging Networking Experiments and Technologies, pages 1 - 12, Rome, Italy, December 2009.

[5] C. Casetti, C.-F. Chiasserini, L. Curto Pelle, C. Del Valle, Y. Duan and P. Giaccone, *Content-centric Routing in Wi-Fi Direct Multi-group Networks*, arXiv.org, December 2014

[6] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt and Matthias Whälisch, *Information Centric Networking in the IoT: Experiments with NDN in the Wild*, arXiv.org, August 2014

Modeling Quantum Optical Components, Pulses and Fiber Channels Using OMNeT++

Ryan D. L. Engle*, Douglas D. Hodson*, Michael R. Grimaila*,
Logan O. Mailloux*, Colin V. McLaughlin† and Gerald Baumgartner‡
*Air Force Institute of Technology, Wright-Patterson AFB, OH, USA
email: {douglas.hodson, michael.grimaila}@afit.edu
†Naval Research Laboratory, Washington, D.C., USA
‡Laboratory for Telecommunication Sciences, College Park, MD, USA

***Abstract*—QKD is an innovative technology which exploits the laws of quantum mechanics to generate and distribute unconditionally secure cryptographic keys. While QKD offers the promise of unconditionally secure key distribution, real world systems are built from non-ideal components which necessitates the need to model and understand the impact these non-idealities have on system performance and security. OMNeT++ has been used as a basis to develop a simulation framework to support this endeavor. This framework, referred to as "qkdX," extends OMNeT++'s module and message abstractions to efficiently model optical components, optical pulses, operating protocols and processes. This paper presents the design of this framework including how OMNeT++'s abstractions have been utilized to model quantum optical components, optical pulses, fiber and free space channels. Furthermore, from our toolbox of created components, we present various notional and real QKD systems, which have been studied and analyzed.**

I. Introduction

Modeling and Simulation (M&S) is often used as an efficient means to understand complex systems and their dynamics, as the model developer (i.e., a researcher or tester) is forced to fully understand the behaviors of interest, their inputs, and expected outputs in order to accurately simulate the system behavior. By systematically defining and decomposing the complex behaviors of interest, one is able to construct representative models, with varying levels of abstraction and generalization, needed to answer research questions of interest.

In this paper, we present a model and simulation framework to study Quantum Key Distribution (QKD) systems. Quantum Key Distribution (QKD) systems offer the promise to generate and distribute unconditionally secure cryptographic keys [1]. However, real world QKD systems are built from non-ideal components which differ greatly from their ideal counterparts [2], [3]. Our research is focused upon understanding and quantifying the impact component non-idealities have on QKD system performance and security. To achieve our objectives, we have extended the OMNeT++ by developing a quantum key distribution eXperimentation (qkdX) framework to model electrical, optical, and electro-optical components necessary to efficiently simulate complete QKD systems. Using the developed framework, we have designed, constructed, and simulated various QKD systems in order to study non-idealities in real-world implementations [4].

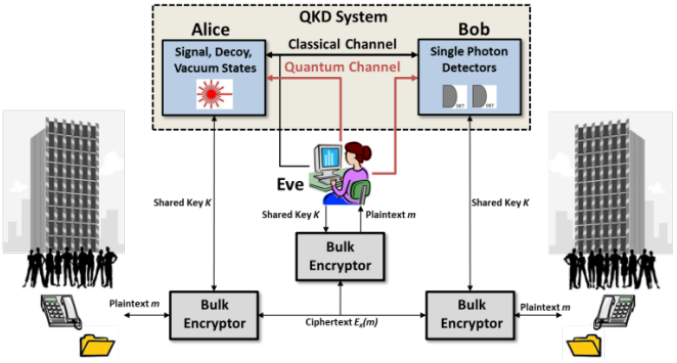


Fig. 1: QKD System Context Diagram

The qkdX framework itself consists of a set of modular components specific to the optics domain, so that QKD systems can be assembled. Using the abstractions for modules, messages and channels, components in the domain of optical systems, such as, beamsplitters, attenuators, optical pulses and fiber channels have been modeled.

This paper presents how we have leveraged OMNeT++ functionality and tailored abstractions to the domain of optical components. We first present a short background on QKD to provide a motivation for this research, followed sections on how the framework is organized, and how OMNeT++'s [5] abstractions have been utilized to model quantum optical components, optical pulses, fiber and free space channels. Finally, we conclude with three specific simulation studies supported by this framework: a polarization-based, prepare and measure BB84 QKD reference architecture, a decoy state enabled QKD architecture and a Measurement Device Independent (MDI) QKD architecture. The results of these studies demonstrate the value of using OMNeT++ as basis for complex system representation, simulation, and analysis.

II. QKD Background

The genesis of QKD can be traced back to Wiesner's idea of encoding messages on photons using polarized photons in conjugate bases to securely communicate information as quantum bits, called "qubits" [6], [7]. In 1984, Bennett and Brassard expanded this idea and proposed the first QKD

protocol, known as BB84, to securely distribute cryptographic key between two parties, typically identified as Alice and Bob [1]. BB84 is a "prepare and measure" protocol where Alice prepares and transmits qubits to Bob who measures them. By exploiting the laws of quantum mechanics and performing statistical analysis of errors during "quantum transmission," QKD systems possess the inherent and unique capability to detect eavesdropping. Thus, theoretical QKD systems can deliver unconditionally secure keys to authenticated parties for use in cryptographic systems. A notional QKD system architecture is shown in Figure 1. An eavesdropper, Eve, is shown attached to the classical and quantum channels linking Alice and Bob. This situation illustrates the sort of contested environment in which QKD systems are designed to operate. An excellent introduction to QKD can be found here [8] and [2].

III. Framework Packages & Organization

The primary objective of qkdX is to enable the rapid and efficient modelling and analysis of current and proposed QKD system implementations using varying levels of model abstraction [2]. From a software point of view, the qkdX framework is a library that defines specific OMNeT++-based modules for optical components (e.g., beamsplitters, attenuators, lasers, classical optical detectors, single photon detectors). The library also defines an abstract message class for which a few concrete pulse types have been defined. Finally, the framework defines a few specific channel types to model the propagation effects associated with optical pulses in fiber cables and free space. For example, an important aspect modeled with channels is the attenuation associated with propagating an optical pulse from one place to another (i.e., one module to another).

As shown in Figure 2, organizationally, OMNeT++ provides the fundamental infrastructure to build and interconnect system components and qkdX defines a set of abstract and concrete system models and components common to many different QKD architectures. End user simulation products crafted to support specific studies and analyzes are then constructed from these components and others (e.g., INET-based [9] components).

IV. Optical Pulses

Much like the flow of classical communication packets, the flow or propagation of quantum optical pulses within the framework are carried by messages, specifically, a specialized message class that manages a pointer to an abstract pulse class as shown in Figure 3. Two concrete types of pulse representations are current defined, CoherentState [10] and FockState [11] - each provides features appropriate to model quantum effects associated with QKD systems.

The base Pulse class defines common attributes associated with all pulses, including wavelength, duration, a global phase which indicates a relative phase offset between it and a reference. Pulses also include polarization information and a characteristic shape. For a Fock state, the shape defines a probability amplitude which is used to determine when a

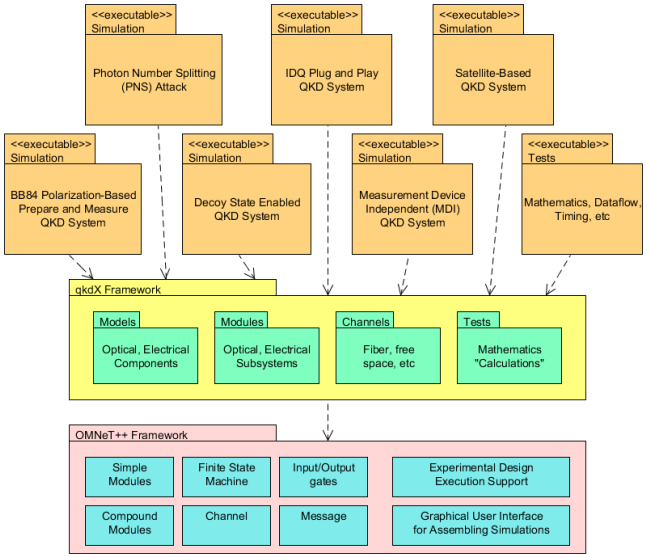


Fig. 2: Package Structure

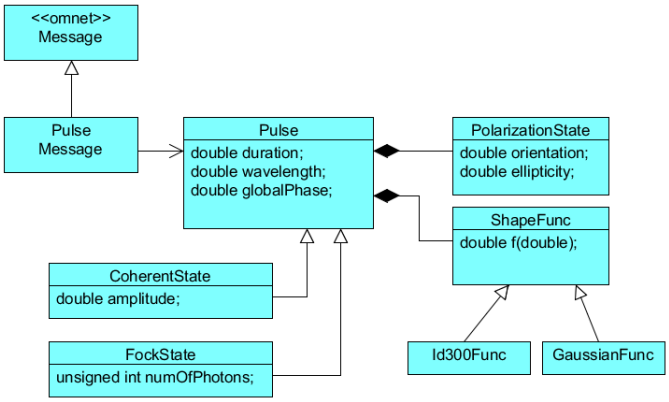


Fig. 3: Pulse Design

photon arrives at a specific device - for a pulse of this type, the number of photons the pulse carries is known, thus its energy is known. For a coherent state representation, the number of photons associated with a pulse is probabilistic, so the shape serves a dual purpose; first to determine the amount of energy contained, and secondly, the time of arrival for zero or more photons. The shape of the pulse is integrated and combined with other parameters (e.g., such as e-field amplitutde) to determine its energy. This energy is divided by the energy of a photon at this wavelength to determine a mean photon number (i.e., MPN). This MPN is used as an input to a Poisson distribution to determine the number of photons present. The pulse shape is then normalized, used to construct a reverse cumulative distribution function, and a uniform random number is drawn to determine the arrival times of the photons.

The shape itself is defined by C++ functor-like class, meaning shapes are defined by objects that encapsulate a single function. The abstract base ShapeFunc class defines the

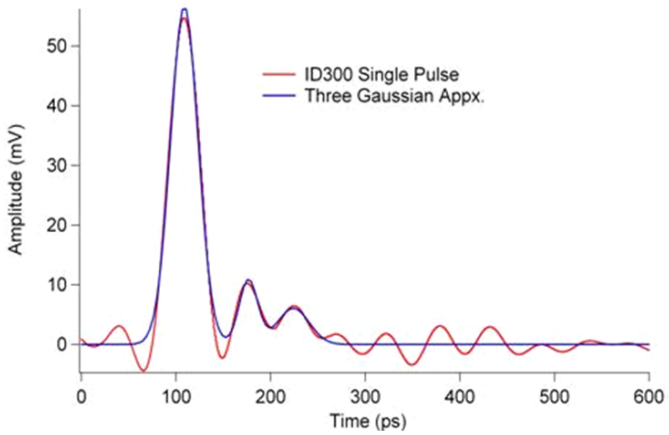


Fig. 4: ID Quantique ID300 [12] 1550nm Laser Pulse

signature for function "f" which is overridden in subclasses. Example subclasses define simple Gaussian shapes as well as measured pulse shapes approximated by multiple Gaussians - an example of this approximation is shown for the ID Quantique ID300 [12] laser (which is used in QKD systems) in Figure 4.

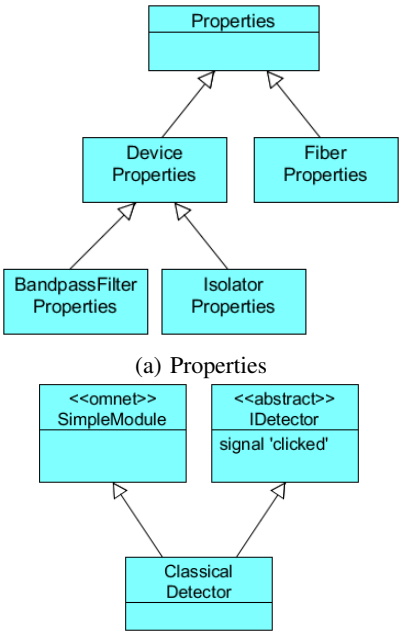
V. Optical Components

Optical components, which are based on simple modules, define the behavior or unique transformations associated with pulses entering and exiting a device through specific optical ports (i.e., module gates). As an example, optical attenuators have two optical ports (i.e., gates), a circulator has three and a beamsplitter has four.

As the framework and software has matured, we have organized the code associated with modules considering four distinct perspectives: 1) as a hierarchal structuring unit with inputs and outputs (i.e., gates) which defines the flow of data (i.e., messages, pulses), 2) as the place where simulation time is known, 3) the place where mathematical calculations are performed, and finally, 4) the place where model *state* is maintained. Furthermore, we view NED file parameters from multiple perspectives as well: as inputs closely associated with the characteristics of a modeled device (i.e., the intrinsic or inherent properties that define the device), initial values for variables (i.e., *state*) that might change during simulation execution, and finally, flags (or switches) to indicate what effects should be modeled or included.

For example, a manufactured beamsplitter of a given type has some inherent properties that are listed on its' specification sheet - these properties are usually read as NED file input parameters. We explicitly differentiate these NED parameters from others, such as boolean flags to turn modeled effects on or off, or set the initial *state* for an optical device (e.g., consider the condition of device as modeled by an `enum` of the values "working," "degraded," or "damaged").

Considering these aspects, we have created a hierarchy of properties as shown in Figure 5a which is used to create "property objects" associated with each modeled optical component. These property objects separate and encapsulate the intrinsic aspects of the device of interest apart from changing state information. They are initialized during module creation by reading NED file parameters (or XML file) and used in conjunction with standalone functions to calculate modeled effects and process optical pulse transformations. Typically, the signature of the standalone functions includes a `const reference` to both its device properties, the optical pulse to be processed, and depending on purpose, *state* information.



(a) Properties

(b) Interfaces

Fig. 5: Component Patterns

This strategy greatly reduces the amount of code written in modules and helps reinforce their role as arbitrators or traffic cops to manage the flow of data, and a few state variables.

In a similar manner to how the INET [9] framework defines abstract interfaces (i.e., contracts), qkdX also defines interfaces for the various optical component categories (e.g., all detectors `emit` a "clicked" signal) as shown in Figure 5b.

VI. Fiber Channels

Modeling fiber cables involves subclassing OMNeT++'s channel abstraction and adding the features needed to account for length, attenuation effects, and even polarization "walk" or "drift" properties. The same approach taken with optical components in terms of creating a property object and standalone functions to process effects was also taken with channels.

VII. Testing

Structuring the code in terms of property objects and standalone *stateless* functions resulted in a significant amount of the codebase not resting (or subclassing) from any OMNeT++ classes. This is a natural result, given that we structured our code so that OMNeT++ modules and channels solely serve

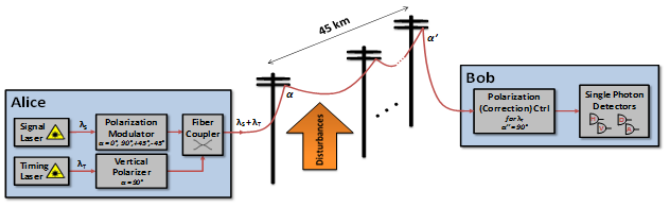


Fig. 6: Polarization Controller Performance Model

the role of traffic cops concerning data flow and the managers of simulation state data. Given this software organization, we found the entire codebase to be more testable.

Using the Simplified Wrapper and Interface Generator (SWIG) [13] utility, it became quite easy to "wrap" the C++ classes and functions with proxy interfaces so they can be imported and accessed from Python [14]. Using Python and IPython [15] in particular, testing the non-time dependent aspects of algorithms was simplified by writing relatively short scripts as compared to say, writing the same test in C++ code. Plotting results is greatly facilitated using various Python support packages such as matplotlib [16]. Another side benefit from this approach became apparent as the team testing component functions did not have to be fluent in C++ to perform this task.

For testing data flow and time dependent aspects of a model, OMNeT++-based simulations have been defined.

VIII. System Simulation Studies

In this section, we briefly describe three different simulation studies that were conducted using the qkdX framework to demonstrate its utility.

We leveraged OMNeT++'s robust data collection and analysis tools to facilitate a deeper understanding of QKD performance-security trades with respect to relationships between quantum phenomenon (e.g., pulse propagation, temperature changes, and physical disturbances) and system-level interactions (e.g., hardware designs, software implementations, and protocols). Further, the ability to rapidly export collected data to statistical analysis programs, such as R [17], provided the ability to quickly summarize large numbers of experiments in an efficient manner.

A. BB84 QKD Reference Architecture

Initially, we defined and modeled a polarization based, prepare and measure BB84 QKD architecture, which we refer to as the reference architecture. Because there was no published examples of QKD systems with sufficient detail necessary to implement a system-level model, we consulted with subject matter experts to define it [2]. This effort was beneficial because it forced us to formalize the behavior of each QKD functional unit.

As we documented each of the functional units, we coded OMNeT++ modules to exhibit the desired behavior. Initially, we constructed this architecture using simple and compound modules that exclusively used the handleMessage() paradigm.

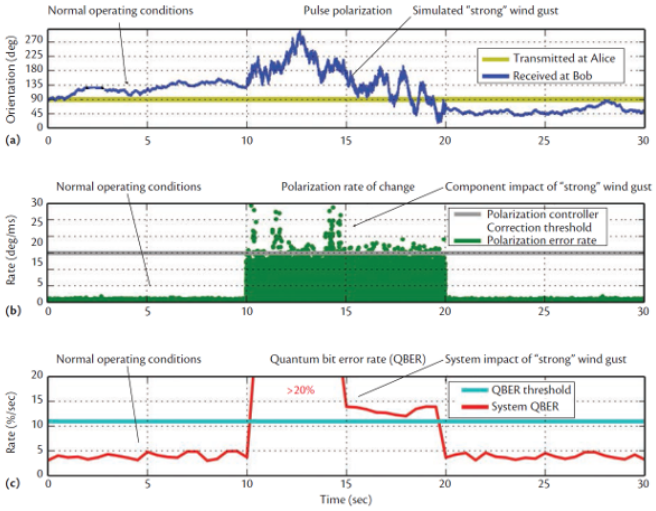


Fig. 7: Polarization Controller Performance Analysis

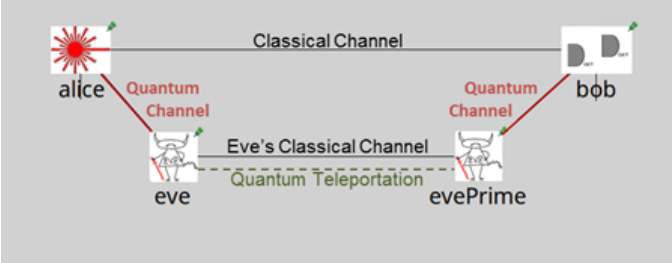


Fig. 8: Decoy State Enabled QKD System Model

However, we found that for our application, in some limited cases, the activity or process-based paradigm was more readable, more understandable, and could more easily be modified.

Our first system-level study using the reference model was to examine the impact that polarization drift played in QKD system behavior. Figure 6 shows an abstraction of the system model used to study the behavior of polarization drift in aerial optical fiber in a real world physical link [4]. The simulation results confirmed experimental results that strong wind gusts in aerial fibers generate sufficient polarization drift to cause system outages. Figure 7 shows a 30-second interval during which initially the polarization is correctable, but then becomes excessive such that it could not be corrected by the polarization controller present within Bob.

B. Decoy State Enabled QKD

As we gained more experience using the qkdX framework, we expanded our exploration into modeling processes and protocols contained in QKD systems related to operational security of the system. One such protocol is the decoy state protocol [18]. In Figure 8, we present a decoy state enabled QKD system model which extended the reference architecture to include the necessary components and behaviors required to perpetrate a Photon Number Splitting (PNS) attack against the QKD system [19]. The PNS attack is conducted by

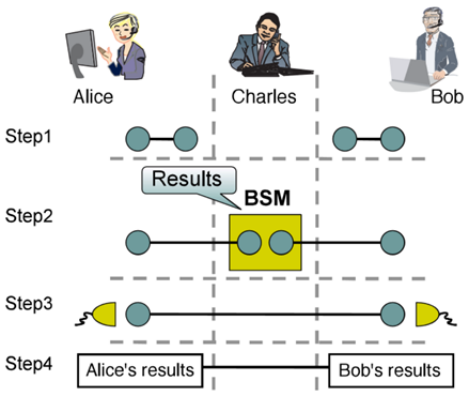


Fig. 9: MDI QKD Process [20]

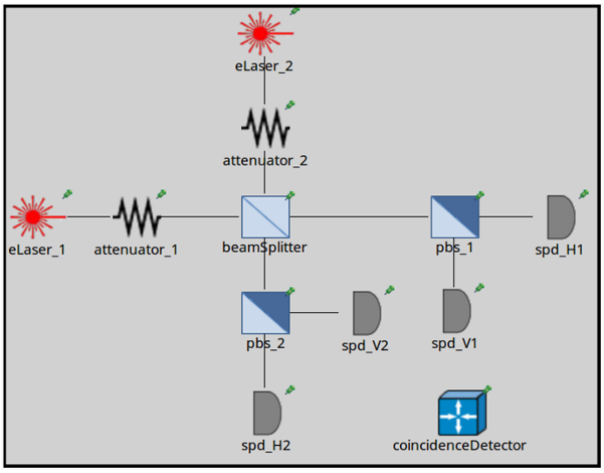


Fig. 10: MDI QKD OMNeT++ Network

an adversarial eavesdropper, Eve, whose purpose is to gain knowledge of the generated secret key. Eve performs the PNS attack by stealing photons from each multiphoton pulse Alice generates while suppressing all single photon pulses [3]. Modeling the PNS attack required implementation of a new optical pulse representation known as the Fock state and required development of notional components. Existing optical component models were modified to handle both weak coherent pulses and Fock states. Additionally, Alice and Bob's individual processors had to be extended to perform analysis of signal and decoy qubit detection statistics to detect the presence of a PNS attack [19].

C. Measurement Device Independent QKD

Detectors are critical components in a QKD system and their operational characteristics greatly impact the performance and security of the system as a whole. One problem identified by QKD researchers was the need to remove detector side channel attacks which can undermine the security of a QKD system. Side channels allow information to leak from the system and can arise from many sources including component imperfections, poor or malicious manufacturers, or the operational environment. This realization led to a new protocol named, Measurement Device Independent (MDI) QKD [20].

In Figure 9, we show a simplified model of a MDI QKD system. This model utilizes components of the qkdX framework to study the Bell State Measurement (BSM) [20]. MDI QKD requires Alice and Bob to individually generate entangled photon pairs. Next, Alice and Bob send half of the entangled photon pair to a third party known as Charles, who performs the BSM and reports his results to Alice and Bob. Then, Alice and Bob perform individual measurements on the retained half of the photon pair. Last, they compare a sample of their results to those reported by Charles. The comparison of these results mitigates attacks on the process by a malicious actor [20].

Figure 10 depicts the OMNeT++ network containing the components used in a MDI QKD system model. Alice and Bob are represented by each eLaser and attenuator pair, while the Bell State Analyzer (BSA) is represented with a beam splitter (BS), polarizing beam splitters (PBSs), and detectors (SPDs). This model required the extension of existing reference model laser capabilities which includes logic to control polarization and detector timing information. Additionally, the modeled beam splitters were extended to represent asymmetric behaviors. Furthermore, the SPDs were enhanced to accurately process interference from multiple pulses (i.e., the BSM) during detection periods known as gates.

Our initial results revealed some deficiencies in our original conceptualization of the MDI QKD model. Instead of attenuating weak coherent pulses down to signal photon levels, we must instead use heralded single photon Fock states to properly calculate the quantum interference effects. Work is currently underway to implement the mathematics necessary to properly account for quantum interference within a BSA apparatus.

IX. Conclusions

In this paper, we presented the qkdX framework, a unique application of the OMNeT++ framework to the domain of modeling optical systems. The motivation to develop this framework stems from a desire to study and better understand the impact of non-idealities on QKD system performance. Using OMNeT++'s module, message and channel abstractions, models of optical components, pulses and fiber cables have been created. We have introduced our approach to modeling these systems using OMNeT++ without including too many details associated with quantum calculations.

We have also presented three QKD system models built leveraging this simulation framework. The system models have been used to efficiently study of QKD systems, protocols, and components.

Acknowledgment

This work was supported by the Laboratory for Telecommunication Sciences [grant number 5743400-304-6448] and in part by a grant of computer time from the DoD High Performance Computing Modernization Program at the Air Force Research Laboratory, Wright-Patterson AFB, OH.

DISCLAIMER

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Air Force, the Department of Defense, or the U.S. Government.

REFERENCES

[1] C. Benett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, 1984.

[2] L. Mailloux, J. Morris, M. Grimaila, D. Hodson, D. Jacques, J. Colombi, and G. Baumgartner, "A modeling framework for studying quantum key distribution system implementation non-idealities," *IEEE Access*, 2015.

[3] G. Brassard, N. Lutkenhaus, T. Mor, and B. Sanders, "Limitations on practical quantum cryptography," *Physical Review Letters*, vol. 85, no. 6, pp. 1330–1333, 2000.

[4] L. Mailloux, M. Grimaila, D. Hodson, G. Baumgartner, and C. McLaughlin, "Performance evaluations of quantum key distribution system architectures," *IEEE Security and Privacy*, vol. 15, no. 1, pp. 30–40, 2015.

[5] "OMNeT++ Discrete Event Simulator," https://omnetpp.org, accessed: 2015-06-28.

[6] S. Wiesner, "Conjugate coding," *ACM Sigact News*, vol. 15, no. 1, pp. 78–88, 1983.

[7] B. Schumacher, "Quantum coding," *Physics Review A*, vol. 51, no. 4, pp. 2738–2747, 1995.

[8] M. Grimaila, J. Morris, and D. Hodson, "Quantum key distribution: A revolutionary security technology," *The Information System Security Association (ISSA) Journal*, pp. 20–27, 2012.

[9] "INET framework," https://inet.omnetpp.org, accessed: 2015-06-28.

[10] "Coherent states," https://en.wikipedia.org/wiki/Coherent_states, accessed: 2015-06-28.

[11] "Fock or number state," https://en.wikipedia.org/wiki/Fock_state, accessed: 2015-06-28.

[12] "ID300 series sub-nanosecond pulsed laser source datasheet," http://www.idquantique.com/images/stories/PDF/id300-laser-source/id300-specs.pdf, accessed: 2014-03-05.

[13] "SWIG simplified wrapper and interface generator," http://www.swig.org, accessed: 2015-06-28.

[14] "Python," https://www.python.org, accessed: 2015-06-28.

[15] "IPython," http://ipython.org, accessed: 2015-06-28.

[16] "matplotlib plotting package for python," http://matplotlib.org, accessed: 2015-06-28.

[17] "The R Project for Statistical Computing," http://www.r-project.org, accessed: 2015-06-24.

[18] H.-K. Lo, X. Ma, and K. Chen, "Decoy state quantum key distribution," *Physical Review Letters*, vol. 94, no. 230504, 2005.

[19] R. Engle, M. Grimaila, L. Mailloux, D. Hodson, G. Baumgartner, and C. McLaughlin, "Developing a decoy state enabled quantum key distribution model," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2015 submitted.

[20] F. Xu, M. Curty, B. Qi, and H.-K. Lo, "Measurement-device-independent quantum cryptography," *IEEE Journal of Selected Topics in Quantum Electronics*, 2014.

Integration of the Packetdrill Testing Tool in INET

Irene Rüngeler
Münster University of Applied Sciences
Dept. of Electrical Engineering and Computer Science
Bismarckstrasse 11
D-48565 Steinfurt, Germany
Email: i.ruengeler@fh-muenster.de

Michael Tüxen
Münster University of Applied Sciences
Dept. of Electrical Engineering and Computer Science
Bismarckstrasse 11
D-48565 Steinfurt, Germany
Email: tuexen@fh-muenster.de

***Abstract*—Google released in 2013 a script-based tool called `packetdrill`, which allows to test transport protocols like UDP and TCP on Linux and BSD-based operating systems. The scripts defining a test-case allow to inject packets to the implementation under test, perform operations at the API controlling the transport protocol and verify the sending of packets, all at specified times. This paper describes a port of `packetdrill` to the INET framework for the OMNeT++ simulation environment providing a simple and powerful method of testing the transport protocols implemented in INET.**

I. Introduction

The complexity of software involves a great vulnerability to bugs. As a consequence, testing becomes more and more vital, but also time consuming. The possibility to automate tests is therefore a must.

The INET framework includes a test suite [1], that features regression, module and validation tests. The user can define tests that are validated via fingerprints or the comparison of parts of the output with a regular expression. These tests can run individually or in a batch job. The test suite is very well suited to test new features for a protocol that expect a certain observable output, but it is hard to test scenarios where you have specific interactions with the applications or specific patterns in packet reception.

As our focus lies on transport protocols we needed a testing tool with special emphasis on the transport layer and its interaction with the application via the socket Application Programming Interface (API). Our preferred transport protocol is Stream Control Transmission Protocol (SCTP), but a more versatile test tool that can be used to also test other transport protocols like User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) would be of broader interest.

Cardwell [2] published the open-source scripting tool `packetdrill`, that "enables testing the correctness and performance of entire TCP/UDP/IP network stack implementations, from the system call layer to the hardware network interface, for both IPv4 and IPv6" [3]. `packetdrill`, developed at Google, is what we were looking for. It combines the following features:

- It is possible to call system calls and compare the expected packets with the real packets created by the network stack.
- The timing is supervised and wrong timing is a reason to fail a test.
- Packets with unsuitable parameters can be injected in the stack and the reaction of the implementation observed.
- A suite of test scripts can be designed and used for regression.

But, it is only suitable to test kernel stacks, since it explicitly uses the socket API and deals with real interfaces.

Our intention was therefore to port `packetdrill` to the INET framework in order to benefit from the same testing features as the kernel implementations.

Yet, another advantage would be to compare the simulation with the Linux and BSD kernel implementations and even use the same test scripts.

To achieve these goals, support for TUN interfaces was added to the INET framework. These interfaces are used to handle the packets for testing. Furthermore, the `packetdrill` application has to be ported from a procedural to an object oriented program. Finally, SCTP support has to be added to support all transport protocols currently being implemented in the INET framework. We are in the process of contributing this work to the public INET repository and hope that our code will be accepted soon.

This paper describes our approach and is structured as follows: In Section II the features of `packetdrill` as implemented by Google will be introduced. The porting of this tool, its architecture and necessary new components in INET, are explained in Section III. Section IV concludes the paper and gives an overview of the future work.

II. The Google Packetdrill Testing Tool

The Google `packetdrill` testing tool was created to ease the process of testing during development, debugging and regression [2]. The scripts can be tailored to test exactly the packet flow that is needed being able to leave out parameters that are not of interest for the test. The script syntax is similar to that of *tcpdump* for packet descriptions and *strace* for system calls. *tcpdump* is a tool for capturing and analyzing packets which is available on most Unix systems. *strace* is a debugging tool showing issued system calls available on Linux systems.

Like every other test tool `packetdrill` has a System Under Test (SUT) and an application that handles the tests. For each test a script is written that features the expected message flow.

```
 0.000   socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
 0.000   setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
 0.000   bind(3, ..., ...) = 0
 0.000   listen(3, 1) = 0
 0.100 < S  0:0(0) win 32792 <mss 1460, sackOK, nop, nop, nop, wscale 7>
 0.100 > S. 0:0(0) ack 1 <...>
 0.200 <  . 1:1(0) ack 1 win 257
 0.200   accept(3, ..., ...) = 4
 0.300 < F. 1:1(0) ack 1 win 260
+0.000 >  . 1:1(0) ack 2
 0.320   close(4) = 0
+0.000 > F. 1:1(0) ack 2
+0.000 <  . 2:2(0)  ack 2 win 32792
```

Fig. 1. Example of a `packetdrill` script

The script in Figure 1 shows a TCP server as the SUT, which accepts a TCP connection. Then the client closes the connection. Initially a TCP socket is opened. Then a socket option is set, the socket is bound and `listen` is called. The server awaits a `SYN` from the client and answers with a `SYN-ACK`. The handshake is concluded by an `ACK` from the client. The characters `'<'` and `'>'` denote the direction of the packet. `'<'` indicates that the packet is injected, i. e. it is sent via the link layer towards the stack being tested. This way mimicks the client's reaction. It is possible to send packets with wrong values in order to test the SUT's behavior. To be able to inject a packet, a virtual network TUNnel (TUN) interface is used. It is automatically opened by `packetdrill` and has a direct connection to the application. Packets starting with `'>'` are created to be compared to the real packets being sent by the stack being tested. The real packets are created by the stack either as a reaction to a system call or a message from the client. The outgoing real traffic is read using the `libpcap` [4] and compared to the expected packets. For example, the outgoing `SYN-ACK` is created by `packetdrill` and stored to be compared to the sniffed `SYN-ACK`. In the course of the script the FIN bit is sent by the client and the server invokes the `close` call.

When `packetdrill` is started the script is parsed, the packets are created and stored either to be injected at a specified time or to be compared later. Then a second thread is started, in which the events (system calls and packets) are processed one by one according to their time stamps.
There are six ways to set the time a packet is expected to be processed. It can be absolute, relative (e.g. $+0.1$) or in a time range (e.g. $0.1 \sim 0.2$). Wildcards can be used or time boundaries for blocking system calls specified. The expected time of an outgoing packet is compared with the real time of a live packet. When the difference is greater than a configured time tolerance, the test has failed. In the case of injected packets the time stamp sets the injection time.

`packetdrill`, as described in [2] and [3] can be used in two different modes, the local and the remote mode. In the local mode only one host is needed, whereas in remote mode two hosts are needed. In local mode a logical TUN interface is used allowing the testing of the transport and network stack, but not capturing any effects (like offloading checksum computations, segmentation and reassembly, etc) a real network interface card and its driver produce. However, in both modes the same logical network setup is used which is depicted for IPv4 in the following Figure 2.

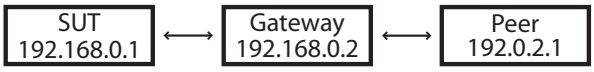


Fig. 2. IPv4 Address Configuration for `packetdrill`

The SUT is using a private IPv4 address and communicates with a peer using an address reserved for testing via a gateway in the same network as the SUT.

Besides system calls and packets Google `packetdrill` supports shell and python commands. To configure the SUT using *sysctl* or assess the state of the machine, a shell command can be included in the script. To print information or use asserts, python snippets are added to the script.

To select one of the supported network layer protocols IPv4 or IPv6 a command line flag can be set. Thus, there is no need to change the script.

In addition to the transport protocols UDP and TCP, we implemented support for SCTP.

III. Porting packetdrill to INET

As the transport layer, especially SCTP, is our main research subject `packetdrill` is the ideal testing tool for us. The first idea was to only add SCTP support to Google `packetdrill`. But as we always keep the SCTP implementation in INET up-to-date with kernel SCTP, we decided to port `packetdrill` to INET. Our goals were to use the same scripts for the kernel version and the simulation and to reuse as much code as possible. As Google released `packetdrill` under the GPL2 integration of it in INET is not a problem.

A. Addition of TUN Interfaces

The TUN interface, as shown in Figure 3, is integrated like any other interface to the NodeBase node. It is activated via a parameter in the omnetpp.ini configuration file. The specific feature of this interface is its direct connection to an application. Thus, traffic arriving from the network layer at the TUN interface is diverted to an application to be processed instead of to another host. To get a better overview of the packets traversing the interface, the pcap recorder can be

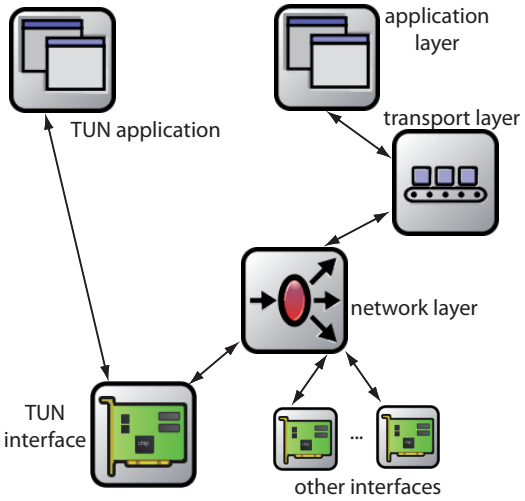


Fig. 3. TUN Interface and its Application

used to trace the traffic. The TUN interface has already been accepted into the integration branch of the inet-framework.

Besides this general behavior of a TUN interface, we had the specific requirement to supervise the traffic going up and down the protocol stack because we wanted to inject packets, i.e. going up, and compare the real traffic to the expected packets, i.e. going down. The solution was an application for both the TUN interface and the transport layer as shown in Figure 4. The new PacketdrillHost has gates to the TUN interface and to UDP, TCP and SCTP on the transport layer.

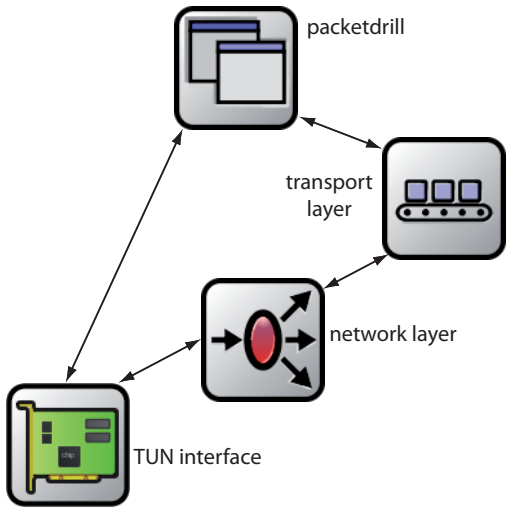


Fig. 4. `packetdrill` Host

B. Software Adjustments and Enhancements

Central parts of Google `packetdrill` are the parser and the lexer that verify the script and generate the list of events to be processed in the course of the test. We did not want to reinvent the wheel. Therefore, we decided to keep as much of this mechanism as possible. But still we had to adjust the parser to the requirements of INET. As Google `packetdrill` is written in C while INET is implemented in C++, a lot of new classes had to be designed to substitute the structures and other data types used in Google `packetdrill`.

To meet the timing requirements of the tests, a strict sequence of the events had to be established. As we only have one thread working in INET, events cannot be processed in parallel. The relative time of an event can only be determined, after its predecessor has been processed. Event timers have to be scheduled to start the sending or receiving of packets at the correct time. The events are numbered not to allow the processing of an event before the predecessor is finished. When simulated packets are travelling down the stack they do not require simulation time. Therefore, it is possible to expect a packet at `packetdrill` via the TUN interface in zero seconds. But still, different arrival times are possible, if, for instance, retransmission times are expected. If the timing behavior is not in the focus of the test, wildcards can be specified to make clear that any time is accepted.

Outgoing packets have to be created and stored until the corresponding live packet arrives via the TUN interface. After `packetdrill` has verified that the live packet arrived in the expected time range, the live and the stored packet have to be compared starting at the IP layer. Injected packets have to be filled exactly, i.e. a complete IP datagram has to be defined in the script, because the packet is sent directly up the stack mimicking the arrival of a packet from a remote host. The definition for outgoing packets is not as strict. Parameters that are not of interest for the test, may be omitted. Yet, it must be possible to compare any parameter that could be used in the protocol. If a parameter is not equal in both packets, a termination exception is thrown and the test is finished.

As mentioned before, system calls trigger commands that are sent from the application layer to the transport layer where they initiate a predefined reaction. The communication between these layers is defined in the socket API of the respective transport protocol. A frequently used system call is *setsockopt()*, which sets options for the created socket (see script in Figure 1). Up to now it was not possible for TCP and SCTP in INET to set parameters via a system call. Options were mostly declared in the .ned files of the application and the modules of the transport protocol and set in the configuration file. But as the scripts for real implementations should be usable in the simulation, a mechanism had to be realized to hand socket option down from the application to the socket and up from the transport level to the socket in order to be able to overwrite predefined values. The actual values are then stored in the main module of the transport layer.

For UDP and TCP the grammar in Google `packetdrill` was taken, but for SCTP [5] a new grammar with new rules had to be written. SCTP's packets consist of a common header and different chunk types, each with a header and parameters of its own. All possible combinations had to be respected in the parser and the lexer and for all chunks functions had to be added to compare the content.

Once the main structure of a protocol is integrated it will not be difficult to add new protocol features, like new chunk types.

C. Test Cases

As the scripts are embedded in the INET test suite, their specification follows the same rules as other module tests. The

packetdrill tests always consist of a script, a .ned file, a routing file and a configuration file like the following.

```
%description:
SUT is server. It accepts a connection. The peer
closes the connection.
%#--------------------------------------------
%inifile: omnetpp.ini

[General]
network = PacketDrillTcp
debug-on-errors = true
ned-path = .;../../../../../src;../../lib

**.scriptFile="../../lib/openPassive.pkt"
**.pdhost.numTcpTunApps = 1
**.hasTun = true

**.startTime = 2s

**.pdhost.routingFile = "../../lib/pdhost.mrt"
**.pdhost.numPcapRecorders=1
**.pdhost.pcapRecorder[0].pcapFile
   ="openPassive.pcap"
**.pdhost.pcapRecorder[0].moduleNamePatterns
   ="tun"
**.pdhost.pcapRecorder[0].sendingSignalNames
   ="packetSentToUpper"
**.pdhost.pcapRecorder[0].receivingSignalNames
   ="packetReceivedFromUpper"

**.pdapp.dataTransferMode = "bytecount"
**.tcp.mss = 1460
**.tcp.sackSupport = true
**.tcp.windowScalingSupport = true
**.tcp.windowScalingFactor = 6
**.tcp.advertisedWindow = 29200
**.tcp.useDataNotification = true

%#--------------------------------------------
%not-contains: test.out
Packetdrill error:
%#--------------------------------------------
```

The %description characterizes the test that is configured in the following %inifile. The script file is specified, the protocol, the routing file and the parameters for the pcapRecorder. After the protocol specific parameters the condition for a successful test is stated. The cause for the above mentioned termination exception always starts with "Packetdrill error:" and can thus be used as an error indication.

D. Limitations of the simulation in relation to Google packetdrill

Looking at Google packetdrill as a model for the INET version, not all features have been implemented. The following limitations apply currently:

- Remote mode unsupported
  The remote mode allows also for testing any interactions of the network interface card with the transport layer. Since the network interfaces in the INET framework do not perform any kind of offload or other interaction with the transport layer, the support of the remote mode was not a priority.
- Python snippets, shell code and command line arguments unsupported
  In Google packetdrill command line arguments are interpreted and python snippets can be included as well as shell code. In INET the command line arguments could be realized by adding configurable parameters to the .ned files and thus configure them in the test file. It would probably mean a major enhancement to INET to allow the interpretation of python snippets and shell code. As we have no use case for this feature, we did not implement it initially.
- Blocking system calls unsupported
  Since INET does not support blocking system calls there was no need to implement them.
- *getsockopt()* unsupported
  The system call *getsockopt()* as a counter part to *setsockopt()* can be used to query socket options. Initial test cases had no need for *getsockopt()* support. Therefore, is was postponed.
- Explicit address handling unsupported
  In Google packetdrill it is possible to set the addresses of the sender and receiver as part of the event header. This functionality has been postponed until use cases for address handling arise.
- IPv6 unsupported
  Up to now only IPv4 is supported since we are focusing on the transport layer, but IPv6 will be included in a future version, as it becomes more and more important.

IV. Conclusion and Outlook

So far we have ported the main features of Google packetdrill except for those mentioned in the last section. Yet, there are some features that are missing in both Google packetdrill and INET. The support for SCTP has already been implemented in both versions. One of SCTP's outstanding features is multihoming, the parallel use of more than one IP address over one connection. It is desirable to test the handover of traffic from one path to another and the simultaneous use of several paths. Therefore, it needs to be added to packetdrill. After completing packetdrill, future work will focus on the development of test suites for transport protocols being available in the INET framework. These scripts should allow testing the transport stacks in the INET framework and on Linux and BSD based operating systems.

References

[1] A. Varga *et al.*, "INET Test Suite,"
Available at: http://inet.omnetpp.org/TestSuite.html, 2015.

[2] N. Cardwell and B. Raghavan, "Drilling Network Stacks with packetdrill," *USENIX ;login:*, vol. 38 No. 5, pp. 48–52, October 2013.

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

[4] C. Van Jacobson and S. McCanne, "libpcap, Initial public release 1994," *Available at http://www.tcpdump.org*, 2015.

[5] R. Stewart, "Stream control transmission protocol," *RFC 4960*, September 2007.

uIP Support for the Network Simulation Cradle

Michael Kirsche and Roman Kremmer
Computer Networks and Communication Systems Group
Brandenburg University of Technology Cottbus-Senftenberg, Germany
eMail: {michael.kirsche, roman.kremmer}@b-tu.de

***Abstract*—We introduce the ongoing integration[1] of Contiki's uIP stack into the OMNeT++ port of the Network Simulation Cradle (NSC). The NSC utilizes code from real world stack implementations and allows for an accurate simulation and comparison of different TCP/IP stacks and a validation of thereby connected simulation models. uIP(v6) provides resource-constrained devices with an RFC-compliant TCP/IP stack and promotes the use of IPv6 in the vastly growing field of Internet of Things scenarios. This work-in-progress report discusses our motivation to integrate uIP into the NSC, our chosen approach and possible use cases for the simulation of uIP in OMNeT++.**

***Index Terms*—uIP, NSC, OMNeT++, IPv4, IPv6, IoT.**

I. Introduction

TCP/IP stacks are the global enablers of world-wide interconnection and communication of devices and computers. A wide range of stack implementations is available today: from full-scale stacks in Windows and Linux to small-scale stacks for embedded systems like *lightweight IP* (lwIP) [1] and even smaller stacks like *micro IP* (uIP) [1] for resource-constrained devices with 8-bit microcontrollers. Using IP(v6) [2] as the bridging protocol facilitates a seamless interconnection of both resource-constrained and non-constrained devices over the Internet (condensed under the label *Internet of Things*).

Adaptation protocols like 6LoWPAN [3], routing protocols like RPL, and application protocols like CoAP and XMPP [4] use the functionalities of IP. A thorough testing of IP stacks is thus necessary; not only self-contained but also in comparison and interconnection with other deployed stacks and protocols. Testing uIP is usually done both via real-life implementations in testbeds and via simulations with the Contiki simulator Cooja [5]. Contiki is an operating system for embedded (wireless) devices, which contains an implementation of uIP with IPv4 and IPv6 capabilities [6]. Cooja is Contiki's own simulator, enabling a time-accurate simulation of Contiki source code through the use of Java Native Interfaces (JNI) and microcontroller simulators like MSPsim [7] in combination with an event-based simulation core and abstract channel models.

While practical testing is essential before any type of rollout, it is typically only feasible for small scale applications. Testing large scale Internet of Things scenarios, especially when cooperating with Internet-based systems, is important yet hard to manage with large numbers of sensors and actuators, different backbone networks and various Internet-based systems working together. For such cases, simulation is still a standard approach to acquire impressions on run-time characteristics and possible bottlenecks. Testing one IP stack against another is also mandatory to ensure interoperability and to assess and compare a stack's performance (refer to [8]).

The Cooja simulator provides accurate results for code execution times and energy consumption of Contiki-based sensor networks. But it lacks the universality of generic frameworks like OMNeT++ and the ability to simulate diverse communication protocols and technologies. Cooja does allow for a socket-based connection to other computers and real networks with the help of virtual network interfaces and the Serial Line Internet Protocol (SLIP), which can be used to connect nodes in Cooja to other networks and hence emulate border router scenarios where Contiki nodes communicate with non-Contiki nodes over a gateway connection. This enables the emulation of simple Internet of Things scenarios, which strive for an interconnection with Internet-based systems and communication over the Internet. The set-up and management of other communication stacks, devices and protocols outside of the Contiki world lies beyond the scope of Cooja. We think that a universal simulation framework like OMNeT++ provides a more generic environment to simulate scenarios that go above and beyond plain Contiki-based sensor networks. Examining IP stacks from the IoT world is also an important task, we therefore aim to extend INET with uIP support to extend OMNeT++'s applicability for IoT simulations. Extending and using OMNeT++ / INET provides the flexibility and model support to simulate IoT applications that interact over gateways, bridges and backbone networks, while supporting various communication technologies and keeping everything in a single controllable environment.

To provide an adequate simulation of IoT applications, we require accurate simulation models for used protocols. Modeling a complete TCP/IP stack from scratch and validating it afterwards is tedious and failure-prone work. [9] showed that a direct integration of a real-world TCP/IP stack into OMNeT++ is possible. One of the main problems for this type of integration is the constant maintenance necessary to adapt to latest code changes. A different integration approach was proposed by Samuel Jansen with his Network Simulation Cradle (NSC) [10]. The NSC wraps real world stacks to make them available to network simulators with lesser need for maintenance. Jansen's original work for ns-2 has proven to be extendable to OMNeT++ and additional TCP/IP stacks like lwIP. It facilitates a validation of simulated stacks [11], [12]. We therefore favor to integrate uIP into the NSC to simulate uIP-based IoT scenarios with OMNeT++.

[1] Source code available online: https://github.com/michaelkirsche/uip4nsc

The remainder of this report is structured as follows. Further motivational aspects for an integration of uIP into the NSC as well as background information are presented in section II. An overview of our integration approach and validation aspects are introduced in section III. A discussion of possible application scenarios in section IV and a summary of ongoing work in section V complete the report.

II. uIP and the Network Simulation Cradle (NSC)

The investigation of network protocols like uIP is often conducted via simulations to study their behavior and reactions to different parameter settings in large scale environments. However, simulation results are only reliable when simulation models are verified to emulate the real protocol behavior. Validated implementations of protocols in simulators are hence an important precondition for meaningful simulations, while missing validations cast doubts over achieved results.

The Network Simulation Cradle was developed to meet the requirements of an accurate but still feasible simulation of TCP/IP communication stacks. A comparison [10] of common simulation models with testbed measurements showed distinct differences between real world stacks and simulation models and even between different real world stacks interacting with each other. To achieve a realistic behavior in simulations, the NSC utilizes real network stacks from common and available implementations instead of focusing on enhancing already abstracted simulation models. The NSC itself was validated[2] by testing conducted simulations against laboratory setups and a network testbed to ensure a realistic emulation of both protocol behavior (validated by checking the sequence of generated packets) and protocol performance (by comparing achieved goodputs in both simulation and testbed).

In an earlier work by Bless and Doll [9], the motivation was to build on an already validated and proven implementation of the TCP/IP stack as a simulation tool by porting the FreeBSD stack to OMNeT++ and avoid a tedious validation process afterwards. The porting process resulted in several manual code changes, which made maintenance and keeping the stack up-to-date very time-consuming. In addition, the possibility of human errors during the porting process could not be excluded completely. The NSC was developed with the precondition to avoid manual changes of the source code and to facilitate the integration and maintenance of different available TCP/IP stacks. In a nutshell, the approach is to compile the extracted network code of the TCP/IP stacks with support functions and a simulator interface into a shared library, which in turn is used by and linked against the simulation framework.

Running multiple instances of a network stack on a single machine is achieved with the help of a parser, which automatically modifies global variables and their references to enable a static virtualization of C source code [13]. This is necessary because network stacks in operating systems are not designed to allow for multiple instances running at the same time as stacks are simply not re-entrant [14, Sec. 3.4]. Each instance of a stack needs his own version of all global variables to run in parallel. To achieve this, one could compile the stack as a shared library and load multiple instances of it or create a new process for every simulated instance. Both approaches lack efficiency and scalability for simulations of large numbers of TCP/IP instances. Static virtualization by filtering the preprocessed C code is used instead to fulfill NSC's goal of adding as little overhead as possible and refraining from manual code changes. A "globaliser parser" is used to programmatically replace all instances of global variables and all references to them in the C code [15]. The parser's basic premise is demonstrated in the following listings 1 and 2, taken from the uIP code.

Listing 1. Original uIP source code

```
struct uip_conn *uip_conn;  /* current connection */

void uip_process(uint8_t flag)
{ ...
  uip_conn = NULL;
```

Listing 2. After executing the globaliser parser

```
struct uip_conn *global_uip_conn[NUM_STACKS];

void uip_process(uint8_t flag)
{ ...
  global_uip_conn[get_stack_id()] = NULL;
```

Today, the NSC supports the Linux, FreeBSD, OpenBSD and the lwIP stack[3]. While initially developed for the ns-2 simulator, a later port made the NSC and thus supported stacks available for OMNeT++.

Our motivation to provide uIP support for the NSC arises from the fact that uIP was designed for a different class of systems, namely resource-constrained systems. Basic premise for the design was a very low memory footprint [16, Table 13.1] to facilitate the use of uIP on 8- and 16-bit microcontrollers with small (Kilobytes) RAM and ROM sizes. The design premises lead to various well-known performance issues. TCP packets, for example, are sent and acknowledged one at a time. The "one-packet-at-a-time" logic is derived from the fact that uIP uses only a single (and stack traversing) packet buffer to support even the smallest platforms. These and other constrains are documented for example in [1].

While uIP $\rightleftharpoons$ uIP communication is not affected by these constraints, uIP $\rightleftharpoons$ "larger" TCP/IP stack communication can be affected if, for example, the communication partner uses delayed acknowledgments as defined by RFC 1122 [17]. These delayed ACKs effectively constrain the packet transmission. A performance hack for this TCP throughput issue is provided in Contiki [18]. We want to compare uIP's performance, especially when uIP is used together with full-scale TCP/IP stacks from non-constrained Internet-based systems. We also want to evaluate other limitations of uIP in combination with MAC protocols like IEEE 802.15.4, such as the effects of different lower layer fragmentation values on application layer protocol performance. These and other use cases for a NSC with uIP support will be further discussed in section IV.

[2]NSC validation: http://www.wand.net.nz/~stj2/nsc/validation.html

[3]NSC stack status: http://www.wand.net.nz/~stj2/nsc/status.html

III. Integration and Validation

The integration of Contiki's uIP stack follows the example of other supported stacks like the Linux or the lwIP stack. Files containing the network code are integrated into the build process without any modifications. References to unused system functions are implemented as stub functions. The simulator interface utilizes the uIP API while access functions for Contiki's MAC layer are reimplemented to redirect back to the simulator (OMNeT++) interface. Figure 1 depicts the basic principle of the integration with the according function and file names.

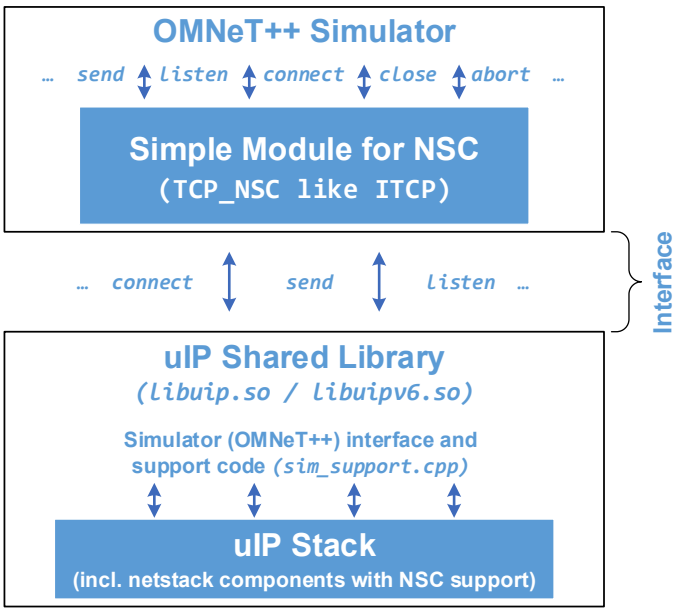


Fig. 1. Integration of uIP in the OMNeT++ NSC module and Interaction between OMNeT++ and uIP via the NSC

The *globaliser parser* changes global variables and their references automatically to arrays. It is integrated as part of the build process of the NSC. Configuring the uIP stack is done via modifications of the `contiki-conf.h` file (refer to listing 3) before compilation time, as it is custom for Contiki. To test different uIP configuration options, it is necessary to recompile the shared library.

Listing 3. uIP configuration options - excerpt from `contiki-conf.h`

```
#define UIP_CONF_UDP                1
#define UIP_CONF_MAX_CONNECTIONS    40
#define UIP_CONF_MAX_LISTENPORTS    40
#define UIP_CONF_BYTE_ORDER         UIP_LITTLE_ENDIAN
#define UIP_CONF_TCP                1
#define UIP_CONF_TCP_SPLIT          0
#define UIP_CONF_LOGGING            0
#define UIP_CONF_UDP_CHECKSUMS      1
#define UIP_CONF_BUFFER_SIZE        400
#define PACKETBUF_CONF_SIZE         400
...
// definitions of Contiki netstack components
#define NETSTACK_CONF_MAC           nsc_mac_driver
#define NETSTACK_CONF_RDC           nullrdc_driver
#define NETSTACK_CONF_RADIO         nullradio_driver
#define NETSTACK_CONF_FRAMER        framer_nullmac
#define NETSTACK_CONF_NETWORK       uip_driver
```

The customized network stack (netstack) components are a peculiarity of Contiki. Contiki's network stack allows to exchange modules and drivers like the MC protocol, the framer or the radio driver. Customized drivers were introduced to interconnect the uIP stack and dependent modules with the simulation support code and the NSC.

Future releases of Contiki containing new and changed versions of uIP should work without modifications as long the basic uIP API is not changed and configuration options are reviewed for possible changes. We switched during our integration between the Contiki releases 2.6, 2.7, and the Github development version. The latter one required changes of the configuration options in the `contiki-conf.h` file and adaptations of the support code for the communication between the shared uIP library and OMNeT++ through the NSC. These changes were necessary because uIP was extended with a socket API to ease maintenance and code development inside Contiki. The support code was changed to use the new socket API of uIP. As soon as the new Contiki version 3.0 will be released, we plan to test the integration of this latest release of uIP into the NSC to determine if our integration process is valid for newer releases too.

The validation process of new stacks is suggested to consist of two steps [19]. In the initial testing, the stack should be able to perform basic operations and then be expanded to communication tests with multiple instances. The goal is to ensure that code extraction and application of the *globaliser* produced a functioning simulation model, which still needs to be fully validated [19]. The already supported stacks can be utilized for additional functionality and interoperability tests.

For the validation of a new network stack, a simulation and testbed comparison is suggested [19]. The NSC interface is able to trace and save packets in the `libpcap` file format. These packet traces can be directly compared to `tcpdump` traces from an equivalent real-world test setup. The goal is to achieve realistic behavior in the simulation compared to the real world in different use case scenarios.

IV. Applications and Use Cases

The NSC has been proven to be useful for performance testing and has shown a wide variation between TCP/IP implementations and significant differences from simplified models [11]. The ability to simulate with real implementations can be used both as a validation and an analyzing tool. We will further discuss three use cases for both fields of applications.

As we are interested in simulations of complex Internet of Things scenarios, where Internet-based systems work together with sensor networks running with uIP, we first want to use OMNeT++ with the NSC to conduct interoperability tests between uIP and other real world TCP/IP stacks. As it was pointed out in section II, uIP has well-known weaknesses when interacting with full-scale TCP/IP stacks. Certain issues can be omitted or mitigated by selecting suitable configuration parameters for uIP. Examples here are the fragmentation limits, buffer sizes, and timing options for TCP and IP itself. An example IoT scenario for our evaluation is depicted in figure 2.

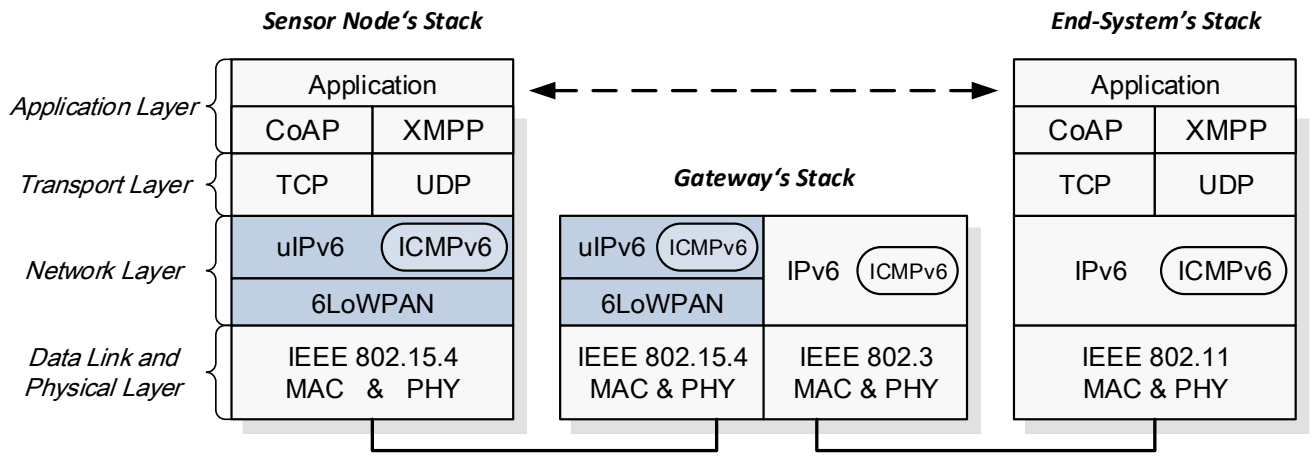


Fig. 2. Internet of Things Evaluation Scenario

In combination with our previously introduced 6LoWPAN simulation model for OMNeT++ [20], we want to evaluate the performance of both uIP and 6LoWPAN fragmentation when resource-constrained sensor nodes communicate with gateways and Internet-based systems (cp. figure 2). Fragmentation on the lower layers of the uIP stack is an interesting research field, as the exact fragmentation border depends on the actually used sensor node and its transceiver [21]. Due to different fragmentation limits, the supported IP payload sizes for uIPv4 / uIPv6, and in general for Contiki-based systems, vary for different sensor nodes. We want to review what influence fragmentation in combination with problems like the "one-packet-at-a-time" principle has on actual IoT application protocols. This goes beyond a simple performance analysis, as packet fragmentation can also affect protocol operations and timer behavior. The ability to trace the protocol operations with `libpcap` traces directly captured from the NSC will hopefully help us to identify issues here.

A third use case is the extensive testing of interoperability issues. The scenario from figure 2 does depict a case where a resource-constrained device communicates with a non-constrained device (i.e., a gateway in this case). Interoperability testing also needs to consider scenarios with heterogeneous sensor nodes. A study [8] of interoperability between Contiki's uIP and tinyOS' embedded IP stack *blip* [22] showed the need for performance tests and additional interoperability and compliance tests for all components of a TCP/IP stack. The authors of [8] discovered that the packet reception ratio decreased for as much as 10% when switching from homogeneous to heterogeneous deployments of sensor nodes with Contiki and tinyOS combined. We hope that future simulations of such heterogeneous scenarios will enable further insights on the actual performance of uIP when used in combination with other stacks.

To summarize, simulating scenarios with the NSC will generally allow us to perform rapid, controllable, and repeatable experiments, which is why we strive to extend the NSC with uIP support to enhance OMNeT++-driven IoT simulations.

V. Ongoing and Future Work

We are currently in the testing phase (cp. section III) for the integration of uIPv4 into the NSC. A thorough testing of the code extraction and *globaliser* operations is in progress. Next steps will be a validation with the help of a real-world testbed based on several sensor nodes running Contiki applications. The integration of the IPv6 part of uIP is also currently in progress. Further testing of uIPv6 will follow once the uIPv4 integration is finalized and validated.

A fitting evaluation of our integration approach will be possible when the upcoming Contiki version 3 is released. Updates to the uIP internals are planed for this release. We hope that this future release can be used with the extended NSC without any code changes (with the exception of added and changed configuration parameters).

Future work could also include the integration of additional stacks from the Internet of Things field into the NSC and thus into OMNeT++. There are several other stacks available, from tinyOS's blip stack [22] to the stacks of RIOT [23] and other WSN operating systems, which could be reviewed for further integration and testing with the NSC.

References

[1] A. Dunkels, "Full TCP/IP for 8 Bit Architectures," in *Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003)*. ACM, May 2003, pp. 85–98. [Online]. Available: http://dx.doi.org/10.1145/1066116.1066118

[2] J. Hui and E. Culler, "Extending IP to Low-Power, Wireless Personal Area Networks," *Internet Computing, IEEE*, vol. 12, no. 4, pp. 37–45, August 2008.

[3] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," IETF, RFC 6282, September 2011.

[4] M. Kirsche and R. Klauck, "Unify to Bridge Gaps: Bringing XMPP into the Internet of Things," in *Proceedings of the Work in Progress Session of the 10th IEEE Conference on Pervasive Computing and Communication (PerCom 2012)*, March 2012.

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

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

[7] J. Eriksson, A. Dunkels, N. Finne, F. Oesterlind, and T. Voigt, "MSPsim – an Extensible Simulator for MSP430-equipped Sensor Boards," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, January 2007.

[8] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, J.-P. Vasseur, M. Durvy, A. Terzis, A. Dunkels, and D. Culler, "Beyond Interoperability: Pushing the Performance of Sensor Network IP Stacks," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '11. ACM, 2011, pp. 1–11. [Online]. Available: http://doi.acm.org/10.1145/2070942.2070944

[9] R. Bless and M. Doll, "Integration of the FreeBSD TCP/IP-Stack into the Discrete Event Simulator OMNet++," in *Proceedings of the 36th Conference on Winter Simulation (WSC)*, 2004, pp. 1556–1561.

[10] S. Jansen and A. McGregor, "Simulation with Real World Network Stacks," in *Proceedings of the 37th Conference on Winter Simulation*, ser. WSC '05, 2005, pp. 2454–2463. [Online]. Available: http://dl.acm.org/citation.cfm?id=1162708.1163136

[11] ——, "Performance, Validation and Testing with the Network Simulation Cradle," in *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, ser. MASCOTS '06. IEEE Computer Society, 2006, pp. 355–362. [Online]. Available: http://dx.doi.org/10.1109/MASCOTS.2006.40

[12] ——, "Validation of Simulated Real World TCP Stacks," in *Proceedings of the 39th Conference on Winter Simulation*, ser. WSC '07. IEEE Press, 2007, pp. 2177–2186. [Online]. Available: http://dl.acm.org/citation.cfm?id=1351542.1351928

[13] ——, "Static Virtualization of C Source Code," *Softw. Pract. Exper.*, vol. 38, no. 4, pp. 397–416, April 2008. [Online]. Available: http://dx.doi.org/10.1002/spe.v38:4

[14] C. P. Mayer and T. Gamer, "Network Simulation Cradle - Final Report for COMP420Y," [online] http://wand.net.nz/pubs/199/pdf/report-final.pdf, WAND Network Research Group, Technical Report, October 2003.

[15] S. Jansen, "WAND Network Research Group: Globaliser," [online] http://research.wand.net.nz/software/globaliser.php.

[16] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann Publishers Inc., 2010.

[17] R. Braden, "Requirements for Internet Hosts - Communication Layers," IETF, RFC 1122, October 1989.

[18] Contiki Doxygen Documentation, "uIP TCP Throughput Booster Hack," [online] http://contiki.sourceforge.net/docs/2.6/a01696.html.

[19] S. Jansen, "Network Simulation Cradle," Ph.D. dissertation, University of Waikato, 2008.

[20] M. Kirsche and J. Hartwig, "A 6LoWPAN Model for OMNeT++," in *Proc. of the 6th OMNeT++ Workshop, co-located with the 6th ICST Conference on Simulation Tools and Techniques (SIMUTools 2013)*, March 2013. [Online]. Available: http://www.omnet-workshop.org/2013/General/TechnicalProgram

[21] R. Klauck and M. Kirsche, "Bonjour Contiki: A Case Study of a DNS-Based Discovery Service for the Internet of Things," in *Proceedings of the 11th International IEEE Conference on Ad-Hoc Networks and Wireless (ADHOC-NOW 2012)*, ser. Lecture Notes in Computer Science (LNCS), vol. 7363. Springer, July 2012, pp. 316 – 329. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31638-8_24

[22] S. Dawson-Haggerty, "Berkeley IP Information," [online] http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip.

[23] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM), Poster Session*, April 2013, pp. 79–80. [Online]. Available: http://dx.doi.org/10.1109/INFCOMW.2013.6970748

Dynamic Index NAT as a Mobility Solution in OMNeT++

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

***Abstract*—Mobility in wireless networks causes a major issue from the IP-addressing perspective. When a Mobile Node (MN) moves to another subnet, it will probably get assigned a new IP address. This causes a routing problem since the MN will not be reachable with its previous IP address known to the other communication party. Real time applications might suffer from connection drops, which is recognized as inconvenience in the currently used service, unless some solution is provided. An approach to maintain session continuity while traversing heterogeneous networks of different subnet addresses is proposed. Here, a cross-layer module is implemented in OMNeT++ with NAT functionality to provide a seamless handover. A proof of concept is also shown with analogy to the Mobile IPv6 protocol provided in INET.**

***Index Terms*—Handover; Network Address Translation; Cross-layer.**

I. Introduction

Wireless communication networks can be categorized as heterogeneous based on different aspects, like having an infrastructure, type of radio access technology, and subnet address. While nowadays communication devices are equipped with multiple interfaces and due to the evolving applications and usage scenarios that require IP connectivity anywhere anytime, switching the connection to another access point (handover) and getting assigned a new IP address is a common communication scenario.

In this paper, we try to handle this issue from the network layer point of view and observe its impact on running applications and the provided Quality of Service (QoS). Nevertheless, we also present our design for vertical handover (VHO) management module, which is however not yet provided with performance analysis. The module manages VHO from above the link layer and between independent wireless interfaces.

For a running session, changing the IP address introduces a routing problem where the MN is not available any more for its communication party, unless a robust mobility solution is deployed. Our proposed approach makes use of the fact that due to the limited address space of IPv4 and the increased number of IP-connected users, Network Address Translation (NAT) became a de facto standard in almost all communication networks. This work implements our concept previously presented in a use case in [1]. Also it contributes to the INET framework of OMNeT++ [2] by implementing NAT operation in network layer with an update mechanism achieved through a cross layer module as will be described in more details later. Performance results are provided in comparison to the known mobility solution (MIPv6) provided in INET. The program code introduced here is a part of a PhD research work and can be available for sharing after the defense of the PhD, to be presented as a VHO protocol contributing to the INET framework.

II. Related work

Next generation network is predicted and being realized to be IP-based, thus service continuity while handover through IP address resolution is a sophisticated research field in wireless communications. The well known mobility protocols Mobile IPv4 (MIPv4) and Mobile IPv6 (MIPv6) [3] [4] are standardized as mobility solutions for the two versions of IP protocols. The idea is to achieve a transparent routing of IP datagrams independently from the mobile host location. This however, introduces delay due to the use of non optimal routing through the home agent, and requires the correspondent node to support the protocol in case route optimization is to be carried out. Several extensions like Fast MIPv6 [5], Hierarchical MIPv6 [6] and Proxy MIPv6 [7] have been proposed that extend the basic MIPv6 to offset its disadvantages. However, IPv6 is not dominant yet in all of our networks neither expected to come fast, and we have to still live with IPv4 for pretty much of time. We assume that the MNs are the initiators of their sessions with the requested application server to start a video stream while the MNs are mobile. However, to have them also reachable for other session initiators from outside, the same update scheme can be deployed to inform the corresponding DNS servers in the area. In this case, private IP addressing is not feasible any more and then, public IP addressing is a necessity. To solve this, the approach of [8] uses a global naming scheme above the IP layer. Although it can be useful for the case of vehicular mobile adhoc communications, it might however harm delay sensitive applications. In relevance to our VHO framework, the one presented in [9] focuses on modeling two types of wireless interfaces and compares two decision algorithms, but a node is not actually able to perform a VHO principally, since the decider node selects a path to send the traffic through, which is a path selection in concept rather than a handover to a new sender/receiver interface of an MN. Our work suggests a quick solution for today's networks of IPv4 with minimum changes to be applied to existing protocols and nodes keeping the problem and the

solution as close to the user as possible. To have adequate performance analysis and comparison, we prove our concept in analogy to MIPv6 that is already provided in INET. At least we do not expect it to provide worse performance than MIPv4, so our comparison should still be valid when we argue upon solutions of different IP versions here. The network setup used is similar to the one presented in [10], which is implemented in INET as well to show MIPv6 functionality. Our paper is in fact a part of bigger research work in progress that proposes a vertical handover cross layer-based engine within the INET framework. So far, in our mobility solution we suggest basically a NAT server capable of dynamically updating its entries based on updates generated by the MN at handover. Namely, we initiate another mobility solution in OMNeT++/INET, try to show the pros and cons of such an approach and explore its feasibility.

III. Addressing Concept

We call our approach Dynamic Index NAT (DINAT) to differentiate it from the well known dynamic NAT. In traditional setups of networks deploying local IP addresses, a gateway with NAT functionality creates a new entry containing global/local IP addresses and ports for any new hosted MN. Any session with a node outside the local network will use the global IP address/es of the gateway. If a handover takes place, a session drop is a major consequence and the MN will then need to restart any session that was running because packets will still be destined to the old IP address of the MN. In our scheme, we assume as a primary step that the different networks are coupled through a single gateway we call GRouter that provides NAT, as depicted in figure 1. This is however a kind of restriction, but we want to proof the feasibility in this stage, and then relax it later in the form of a proxy server available in the cloud as an anchor point for MNs communications.

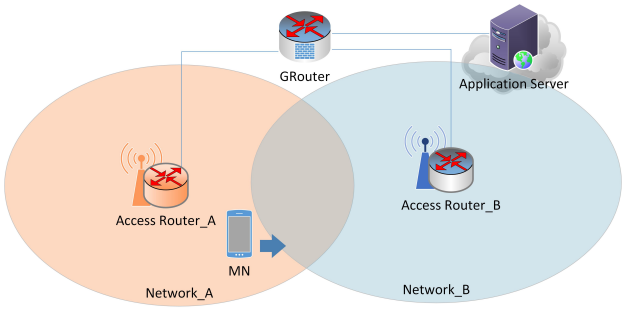


Fig. 1. Network topology

As long as an MN is within a certain network, it is provided with normal NAT functionality by the GRouter. When the IP address of a MN is changed after a handover, a chain of updates between the MN and the gateway of the network takes place. The GRouter receives an update message from the MN including its old and new IP settings to update its NAT entries, so all incoming packets of a running session will be translated to the recently assigned IP address and hence, packets will be re-routed to the new network. Port Address Translation (PAT) is also implemented to support more clients in case of IP overloading in NAT. Figure 2 demonstrates the DINAT mechanism.

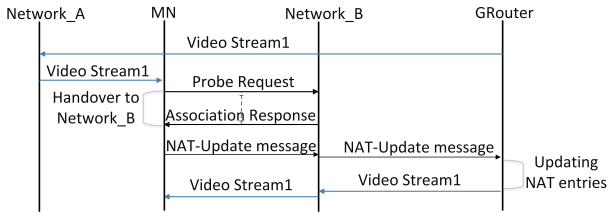


Fig. 2. chart of exchanged messages

IV. Simulated Modules

The suggested approach targets to present a VHO solution within the INET frame work. To realize the design in OMNeT++, functions are added at the Network and Access layers, where both are connected to a cross layer module. We categorize and describe each briefly as follow:

A. Access layer related and cross layer modules

A cross layer module we call the IPCoManager is implemented in the WirelessHost/MN. It manages the wireless interfaces in VHO operations through a module called VHOController. This controls the modules in the wireless interfaces that govern association process with their corresponding networks. Our approach does not care about the details of the differences between the wireless technologies for now but rather, governs their rules in connectivity with respect to the upper layers. A primitive request message will always be sent by the controlling module of an interface (by the Agent module of the WLAN for ex.) to the VHOController to acquire permission to connect. The decision depends on the state of the controller and the result of the decision algorithm inside. Figure 3 illustrates the controller phases of operation from a link layer prospect.

For a stable handover, the controller implements a set of timers that can be setup in the simulation configuration file. For example, when switching ON the user device, the module waits for a specific time to make sure that all surrounding beacons are received and hence selects the best (if no preferred is specified), rather than immediately associating with the first discovered network. This timer is configurable according to the beaconing intervals. A new request is denied if a VHO has just been carried out or a permission is currently waiting for confirmation of association. A time of dual connectivity mode is also possible before releasing the old connection. On a grant of permission to connect and eventually the receipt of Association Confirmation from the concerned interface, the IPCoManager generates NAT update messages that contain old and new IP addresses of the MN before and after VHO respectively. The generated message is then sent to the IPv4 entity for further process. A local version of the message is also generated when switching ON the device and first

selecting a hosting network to set the active interface and gateway at the network layer.

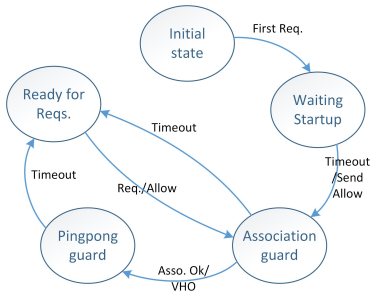


Fig. 3. VHOController state diagram

B. IPv4 related modules

At the host (MN) side, the IPv4 module receives a NAT update message from the IPCoManager in case of handover. It updates its routing table according to the IP address information carried. The NAT update message is then forwarded to its specified destination, which is the GRouter in our case. A local version of the message is received every time the user device is switched ON to set the active interface and gateway in the routing table in accordance to the selected network, and then it is discarded. To add NAT functionality to our GRouter, the IPv4 module provided in INET is deployed. However, to maintain the flexibility of INET, the new version of the IPv4 is deployed through a module interface, since it is required only for the GRouter to provide NAT. A simple module we call NAT-Table is implemented such that, whenever a data packet is received in IPv4, NAT-Table is consulted and the respective IP addresses and Port numbers (for NAT and PAT) are changed accordingly. A new message type and a handle function are defined in IPv4 in relevance to NAT functionality to make it capable of amending its entries dynamically. Whenever the message called NATUpdate is received at IPv4, the information carried by the message (old and new IP/port address) are used to update the specified entry in NAT-Table. Figure 4 shows the added modules.

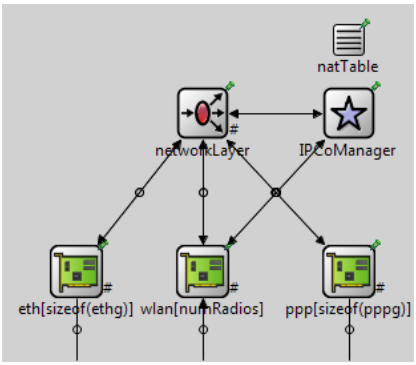


Fig. 4. DINAT modules

V. Simulation Setup and Scenario

The topology of figure 1 demonstrates two networks of different subnet addresses with an MN moving from one network to the other, which is most likely the case for networks of different radio technologies as well. In relevance to the modules description in the previous section, the simulation scenario considers a single wireless interface in the MN. Since this work emphasizes only the addressing problem, radio technology and vertical handover related tasks at the link layer are out of the scope here. The mobile node moves in a linear path from a predefined point inside the coverage area of Network_A, in which it is associated first, towards a specific point within the coverage area of Network_B, where it handovers its connection to. In the meanwhile, it requests the Application Server to run a video stream as a real time application. For proof of concept purpose, the client/MN is enabled to re-request streaming after specific time of stream receipt absence (we call App. Time out). In general, all the IP addresses are assigned statically in the simulation. In different simulation runs, Mobility speed values of 1,2,10 and 20 mps were tested with sending intervals of 5,10,100 and 500 ms at the Application Server using the same packet size. Throughout each run, the MN moved at a constant speed and on a linear path using the mobility module LinearMobility of INET. The simulation time differs for each run according to the mobility speed, however, if equal time was used for all the runs, we would have the MN moving back and forth between the simulation boundaries when high mobility speed is set and hence, more handovers would be experienced, which in turn will produce inconsistent results. Position of all nodes and mobility speed are predefined, so no simulation repetitions were performed for a single set of parameters, but rather for different sets. MIPv6 main parameters like router advertisement intervals are left as in the INET defaults.

To investigate the feasibility of our approach, we compare the impact of handover on two more scenarios in addition to ours that we call DINAT _Case. The second scenario we call Default_Case, which uses absolutely no mobility solution other than a reaction for the stream interruption at the application layer by re-requesting the stream. MIPv6_Case is our third scenario, which applies MIPv6 provided by INET. The three scenarios use the topology shown in figure 1 with the same set of simulation parameters.

VI. Results and Analysis

Since the worse consequence for a handover is the unreachability of the MN, we measure the performance here in terms of packet loss rate in each scenario. This loss rate was measured in multiple simulation runs as an average versus MN mobility speed, where for each velocity value a different set of sending interval values for the Application Server were used. Figure 5 shows the performance of the three scenarios, each ran with a single handover event.

Since the default case has no mobility solution and relies only on the application layer to react after a time of no-receipt of packets is reached, this scenario shows the highest packet

loss as expected and observed through multiple simulation runs. As identification aspect, this is considered as a new stream or session setup at the server. Showing results for a scenario with no mobility solution might not be common but, at least it expresses the amount of improvements the different tested solutions brought. MIPv6 shows a significant improvement over the last. The MN is still defined with its home IP address at the server (and with its new one if we include route optimization procedure) and session continuity is achieved. Our DINAT shows a further improvement, where it is not depending on router advertisements as MIPv6. The handover and re-routing delay depends on the handover delay time, which is link layer related, and on the time starting from trigger initiation at the link layer of the MN till the NAT table in the GRouter is updated. Mobility speed has a huge impact on the performance overall as illustrated, too.

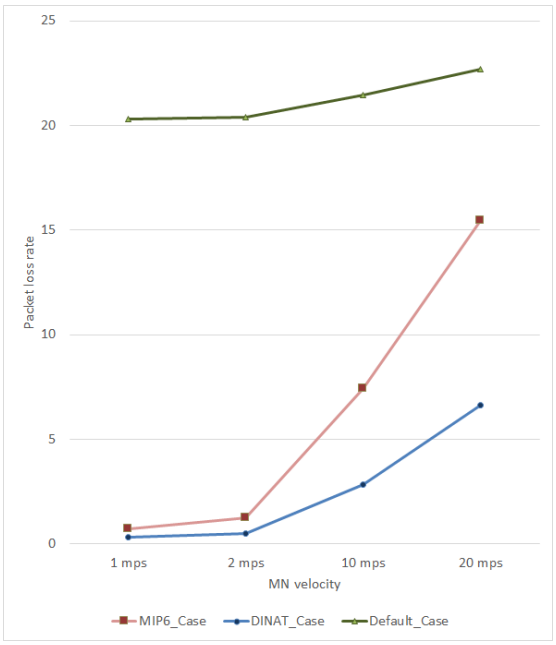


Fig. 5. Packet Loss Rates

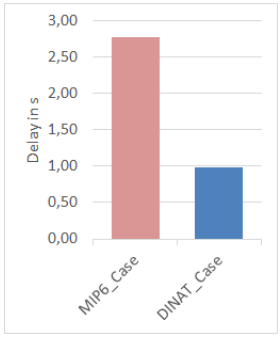


Fig. 6. Delay experienced at the Application layer

If we like to consider the delay sensed by the running application, which we define as the time between the last and the first packets were received before and after the handover respectively, then figure 6 can demonstrate the delay for two of the tested scenarios. There was no point to show that of the Default_Case since it equals to the client App. Time out. However, we can observe less delay experienced by the application in the DINAT_Case.

VII. Conclusion and Future work

We presented a solution to deal with addressing problem as a consequence to handover between different subnets. The proposed approach showed a better performance as it was tested with a video stream traffic. The benefits were shown in terms of packet loss rate and delay sensed at the application layer. Randomness was not considered in the simulation runs, where we focus only on the behavior during the handover and its impact on the running traffic. The well known MIPv6, which is already implemented in INET, was chosen as an opponent approach. Despite it deals with a different category of IP addresses, we expect that MIPv4 would not perform better than MIPv6. Modified versions of MIPv6 are not available for MIPv4, so our comparison will still hold if we want to consider a solution for todays IPv4-based dominant communication networks. In comparison to MIPv6, our approach shows a better performance in term of packet loss during handover, deploys the standard version of IP without the need to add any mobility extension to IP in all the nodes between the MN and the corresponding node, and requires no real IP assignment to the MN. TCP traffic was however not in the scope, since such a type of traffic is less sensitive to handover and packet loss. No modification is needed at application server side, which was intended to be masked from all handover signaling and subsequent events. The NAT update message that is deployed to inform the GRouter about the IP address change however, is not transmitted yet in a reliable manner, so any loss or delay will cause a big negative impact on the overall performance. The network topology tested might not be common in real networks for reasons related to management, quality of service and security, but this work aims to upgrade the DINAT functionality to a proxy server setup globally as an anchor point for the traffic, without the need to couple the network like we did through the GRouter. Users who participate to a certain service will forward their packets to the proxy. Added delay, which may become large then, is an open question on the feasibility of such a setup. The scenarios tested here apply no processing load to the GRouter since only one MN was running. Future work should take into account observing the performance within networks loaded with users. In addition to the mentioned future tasks, the proposed approach is to be expanded to present a package solution for vertical handover. In this case, signaling at the link layer regarding multiple independent wireless interfaces, the decision algorithm concerning handover initiation and network selection, and an address resolution mechanism (like DINAT) represent challenges to be considered.

REFERENCES

[1] A. Al-Rubaye, A. Aguirre, and J. Seitz, “Poster: Address Resolution for Vehicular Communications in Heterogeneous Environments,” in *Vehicular Networking Conference (VNC)*, 2014.

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

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

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

[5] H. Yokota, K. Chowdhury, R. Koodli, B. Patil, and F. Xia, “Fast Handovers for Proxy Mobile IPv6,” in *RFC5949*, 2010.

[6] H. Soliman, C. Castelluccia, K. ElMalki, and L. Bellier, “Hierarchical Mobile IPv6 (HMIPv6) Mobility Management,” in *RFC5380*, 2008.

[7] S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil, “Proxy Mobile IPv6,” in *RFC 5213*, 2008.

[8] S. Schellenberg, P. Begerow, M. Hager, J. Seitz, T. Finke, and J. Schroeder, “Implementation and Validation of an Address Resolution Mechanism using Adaptive Routing,” in *in 27th International Conference on Information Networking (ICOIN)*, 2013.

[9] D. T. C. Cernazanu-Glavan, “OMNeT++ Framework for Efcient Simulation of Vertical Handover Decision Algorithms,” in *8th IEEE International Symposium on Applied Computational Intelligence and Informatics*, 2013.

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

Improvements in OMNeT++/INET Real-Time Scheduler for Emulation Mode

Artur Austregesilo Scussel
Instituto Tecnológico de Aeronáutica - ITA
São José dos Campos São Paulo, Brasil
email: arturscussel@gmail.com

Georg Panholzer, Christof Brandauer, Ferdinand von Tüllenburg
Advanced Networking Center
Salzburg Research Forschungsgesellschaft mbH
Salzburg, Austria
email: {firstname.lastname}@salzburgresearch.at

***Abstract*—In this paper the performance of INET's emulation mode is evaluated. In particular, the focus of the study is on the precision of the delay emulation. It is shown, that this precision is low in INET 2.6 (respectively a later version provided in the integration branch that fixed the crashes of 2.6 in emulation mode). Two errors in the implementation are identified and an alternative configuration for packet capturing is proposed. The performance tests are re-run with such a modified version of the real-time scheduler (which is now included in the recent INET 3.0 release) and it is shown that the responsiveness of the emulation mode and the precision of delay emulation are improved significantly. Finally, the negative impact of the modified capturing configuration is briefly analyzed. Packet loss in the capturing process has deteriorated but in fact is has already plagued the emulation mode of previous implementations and this topic clearly demands for further studies.**

Keywords–OMNeT++; INET framework; Emulation; Real-time simulation

I. Introduction

The work presented here was carried out in the course of an ongoing research project where inter-dependencies between the quality of an electrical energy network and its supporting communication network are analyzed. In the given scenario, power quality metrics are transmitted from a power system simulator to a real-world remote control application which computes new configuration parameters (e.g. setpoints) of the electrical network and send them back to the power system (simulation). The IEC 61850-based communication takes place via an emulated wide-area TCP/IP network.

For the study it is necessary to impair the communication quality (delay, loss, reordering etc.). Typically this is done through a software- (e.g. netem [1] or DummyNet [2]) or hardware-based WAN emulator. However, the next step of the study requires that the real-world communication flow will be multiplexed with many additional simulated flows as produced by simulated energy applications. Ideally, OMNeT++ could be used not just for the simulation but also for the emulation task by utilizing the emulation capabilities of the INET module [3]. It is therefore evaluated if the main requirement - a delay emulation precision in the order of a few milliseconds - can be met by INET's emulation mode. The test environment is *Host1*, a modern PC (Xeon E5-1650v3, 32GB RAM, Intel I350-T4 NIC) running Ubuntu Linux 14.04.

II. Performance evaluation

Several research works based on INET emulation have been published [4], [5], [6], [7], [8]. This however, is already some years back and when we started our work in December 2014 it was even difficult to get emulation mode up and running. The then current stable INET version 2.5.1 crashed upon the first packet arrival and the development version 2.99.0 didn't compile and had the same runtime problems once the compilation problems were sorted out. In fact, INET 2.1 (released in January 2013) was the last version where the emulation mode was running out of the box. It seems that INET emulation has not been widely used in the recent years.

Less than 1 day after we sent a bug report for version 2.5.1 to the OMNeT++ mailing list[1] a fix was provided in the integration branch by Zoltán Böjthe on December 12, 2014. Based on this INET version and OMNeT++ version 4.6 the evaluation as presented in this paper was conducted.

A. Ping to local StandardHost

The first test is based on the `extClient` example that is shipped with INET. The simulation model contains a single StandardHost where 1 external interface (`ext0`) is configured with an IP address of 10.1.1.1. IP traffic targeted to this address is captured on interface `eth1` and injected into the simulation. The real-world `ping` application is used to generate traffic (and receive the replies from the simulated StandardHost). It is run on the machine where OMNeT++ is installed and simply pings the address 10.1.1.1. It thus generates ICMP echo requests and for each ICMP echo reply received it prints the measured round trip time (RTT). OMNeT++ is run from the command line using the `Cmdenv` interface.

In contrast to our expectations, the RTT ranges from approximately 1 ms up to 22 ms with an average of 12 ms. A boxplot of the measured ping RTTs is depicted in the left-hand side ("ping local") of Figure 2. The lower and upper hinges of the boxplot cover the first and third quantiles of the measured values. The whiskers extend to the minimum and maximum of the measured values, respectively.

If, in comparison, the emulation is not used and the ping targets the IP of the local `eth1` interface, the RTTs are always below 1 ms.

B. Ping to router over emulated link

In the second test the simulation model contains 2 routers that are connected by a DatarateChannel with a datarate of 1 Gbps and a delay of 10 ms (in each direction). Additionally, each router has 1 external interface. They are connected to the simulation machine's real interfaces `eth1` and `eth2`, respectively. These interfaces are both connected to remote PCs as shown in Figure 1.

[1] https://groups.google.com/forum/#!forum/omnetpp

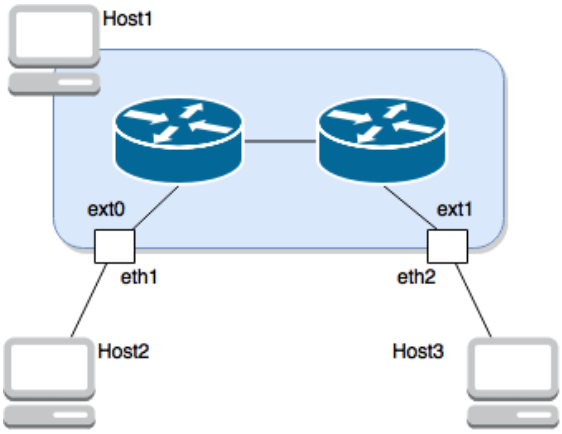


Figure 1. Setup for the performance tests with an emulated link.

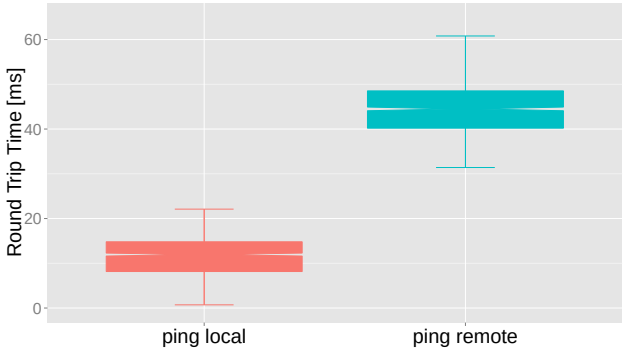


Figure 2. Boxplot for RTTs to local and remote nodes: original code.

Again, a simple ping test is run to obtain an approximate measure of the RTTs. Concretely, *Host2* pings *Host3* over *Host1*. To get a performance baseline first, *Host1* is configured for IP forwarding and OMNeT++ is not run. In this case, the RTTs are permanently below 1 ms. Then, IP forwarding is turned off and the packet forwarding on *Host1* is achieved via OMNeT++/INET. The emulated link between the two routers should increase the RTT by 20 ms and an overall RTT in the order of 21 ms is expected. A boxplot of the measured ping RTTs is depicted in the right-hand side ("ping remote") of Figure 2.

In both tests, the delay emulation is imprecise. The RTTs are significantly higher than what can reasonably be expected and exhibit strong variations. The precision was insufficient for the given requirements of the planned study and we thus analyzed whether (and how) the precision of delay emulation could be improved.

III. Analysis

In emulation mode events are not processed immediately one after the other as fast as possible but instead the processing of events is synchronized with the wall-clock time. This mode is implemented by INET's (soft) real-time scheduler (`linklayer/ext/cSocketRTScheduler.cc`). Additionally, the scheduler enables the integration of real-world network traffic, i.e., real packets can be injected into the simulator and emitted from the simulator onto the real network.

In INET the capturing of packets is implemented via the pcap library [9] which provides a high-level interface to the kernel-level packet capturing facilities for several platforms. Packets are "sniffed" from a real network interface (e.g., `eth0`) and passed to a so called "external" interface of an OMNeT++ host (e.g., `ext0`) by inserting an event into the simulator's future event set (FES). An OMNeT++ Standard-Host/Router module already contains a configurable number of external interfaces. To emit packets to a real network, raw sockets are used.

INET's `cSocketRTScheduler` integrates simulation internal event processing with external packet events (that naturally occur simultaneously) in the following way: when the next simulation event is in the future and the simulation is thus idle (because the event processing must be synchronized with the wall-clock time), the real-time scheduler utilizes this idle time to check if new packets destined to the simulator have arrived on the host's network interface(s).

In search of performance bottlenecks of INET's emulation mode, we realized that this strategy was not implemented correctly. When the next event from the FES occurs at `targetTime`, the scheduler calls `receiveUntil(targetTime)`. Internally, however, this function invokes `receiveWithTimeout()` which makes a blocking call to the pcap library with a fixed timeout of 10 ms (as defined hard-coded in `PCAP_TIMEOUT`). Thus, if the next event is, e. g., 3 ms in the future, the simulator would enter the blocking pcap function and stay there for 10 ms if no traffic destined for the simulator is observed on the host's interface(s). As a consequence, the processing of the next event is approximately 7 ms late. If this event results in emitting a packet on the real network, the packet is sent late.

Before we reported this problem (as we were not done with the overall analysis) it was obviously discovered elsewhere and an issue was filed[2] by András Vargas. A fix was provided by Rudolf Hornig and he additionally resolved the problem that Linux optimized code in the capturing process was *not* activated under Linux.

Another source of additional delays (and delay variations) in the INET emulation mode stems from the way the packet capturing framework is used. In the default mode, a packet is not immediately passed to the application when it is captured. Instead, packets are collected in the kernel until a timeout occurs or the receive buffer fills up (whichever occurs first) and are then passed to the application in one batch [10]. This strategy saves system resources (context switches, system calls) but introduces delays that may be undesirable for the real-time emulation mode. It can be disabled by invoking the `pcap_set_immediate_mode` before a capture handle is activated.

IV. Modified Emulation Code

To evaluate the impact of the above described issues the 3 changes (correct timeout computation; activation of Linux optimized capturing code; activation of the immediate mode via libpcap) were applied and the above tests were re-run. Indeed, the precision of the delay emulation is vastly improved

[2]https://github.com/inet-framework/inet/issues/120

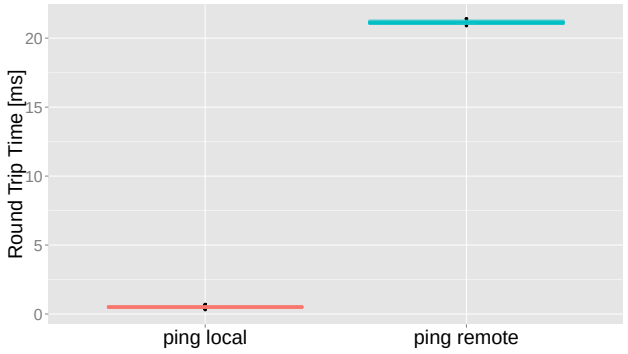


Figure 3. Boxplot for RTTs to local and remote nodes: modified code.

as can be seen in Figure 3. The response times for local pings (ping local) are constantly below 1 ms and the ping RTTs measured over the emulated link (ping remote) are approximately 21 ms. In both cases, the variation of the observed delays is very small.

The tests were once again run against the recently released INET 3.0 version. It includes all the modifications described above. The results are exactly the same as shown in Figure 3.

The immediate mode in the capturing configuration doesn't come without a price. The potential drawback is the increased risk of packet loss in the capturing process since there is no buffer in immediate mode.

We ran several tests with various traffic rates and packet sizes on INET 3.0 with and without immediate mode. Indeed, packet loss in the capturing process could be observed in both versions and the loss was higher with immediate mode. As an example, a test with a generated traffic of 10 Mbit/s and a packet size of 100 Bytes (125 packets/s) yielded a capture loss of roughly 3.4 % (with immediate mode) and 1.6 % (without immediate mode). Other tests showed similar behaviour. These initial tests indicate that the immediate mode increases the packet loss in the capturing process, but packet loss can also happen without it. In any case, this is not acceptable as such a packet loss usually renders an emulation useless. Clearly, this topic requires further studies.

V. Conclusions and Future Work

In this paper we have studied the precision of delay emulation in the INET framework. The study was initiated after we faced unexpected delays with a high variance of delay values when using INET's network emulation mode. One key concern of this issue could be identified as a problem in the timeout computation within INET's real-time scheduler. This problem was fixed by OMNeT++/INET developers. However, this improvement is not sufficient to achieve delay values in a reasonable range. Further analyses of the pcap packet capturing mechanisms in the real-time scheduler has shown that using the pcap immediate mode is considerably reducing the delay.

Evaluations have shown, that with those code modifications both emulation responsiveness and precision of delay emulation could be significantly increased. The proposed code modifications were already integrated in version 3.0 of INET.

Unfortunately, our studies also unveiled packet loss in the packet capturing process ever for low bandwidth/packet rates. This issue is present with and without immediate mode. However, immediate mode increases packet loss due to disabled packet buffering at kernel space. However, this topic could only be covered superficially within this study and urgently needs further investigations. Possible approaches to face these issues would be a multi-threaded implementation of the packet capturing mechanism and/or usage of PF_RING [11], a new type of network socket that affords high-performance capturing under Linux and which is supported in current libpcap versions.

Acknowledgment

This work was carried out in the course of a project that is funded by the Climate & Energy Fund Austria within the "ENERGY MISSION AUSTRIA"' program.

References

[1] S. Hemminger, "Network emulation with NetEm," in Linux conf au. Citeseer, 2005, pp. 18–23. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.1687&rep=rep1&type=pdf

[2] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," ACM SIGCOMM Computer Communication Review, vol. 27, no. 1, Jan. 1997, pp. 31–41. [Online]. Available: http://portal.acm.org/citation.cfm?doid=251007.251012

[3] "INET Framework - INET Framework." [Online]. Available: https://inet.omnetpp.org/

[4] C. P. Mayer and T. Gamer, "Integrating real world applications into OMNeT++," Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, Tech. Rep. TM-2008-2, 2008. [Online]. Available: http://doc.tm.uka.de/TM-2008-2.pdf

[5] T. Staub, R. Gantenbein, and T. Braun, "VirtualMesh: an emulation framework for wireless mesh networks in OMNeT++," in Proceedings of the 2nd International Conference on Simulation Tools and Techniques. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, p. 64. [Online]. Available: http://dl.acm.org/citation.cfm?id=1537696

[6] E. Weingärtner, "Synchronized Network Emulation," 2008. [Online]. Available: https://www.comsys.rwth-aachen.de/fileadmin/papers/2008/sigmetrics-thesis-panel.pdf

[7] S. Corcoradă, "Connecting Omnetpp to virtual Ethernet Interfaces." [Online]. Available: http://www.wseas.us/e-library/conferences/2013/Brasov/MMIC/MMIC-04.pdf

[8] M. Tüxen, I. Rüngeler, and E. Rathgeb, "Interface connecting the INET simulation framework with the real world," in Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops. Brussels, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[9] "TCPDUMP/LIBPCAP public repository." [Online]. Available: http://www.tcpdump.org/

[10] S. McCanne and V. Jacobson, "The BSD packet filter: A new architecture for user-level packet capture," in Proceedings of the Usenix Winter 1993 Technical Conference, San Diego, California, USA, January 1993, 1993, pp. 259–270. [Online]. Available: https://www.usenix.org/conference/usenix-winter-1993-conference/bsd-packet-filter-new-architecture-user-level-packet

[11] "PF_ring | ntop." [Online]. Available: http://www.ntop.org/products/packet-capture/pf_ring/

Realistic, Extensible DNS and mDNS Models for INET/OMNeT++

Andreas Rain Daniel Kaiser Marcel Waldvogel
University of Konstanz, Konstanz, Germany
<first>.<last>@uni-konstanz.de

***Abstract*—The domain name system (DNS) is one of the core services in today's network structures. In local and ad-hoc networks DNS is often enhanced or replaced by mDNS. As of yet, no simulation models for DNS and mDNS have been developed for INET/OMNeT++. We introduce DNS and mDNS simulation models for OMNeT++, which allow researchers to easily prototype and evaluate extensions for these protocols.**

In addition, we present models for our own experimental extensions, namely *Stateless DNS* and *Privacy-Enhanced mDNS*, that are based on the aforementioned models. Using our models we were able to further improve the efficiency of our protocol extensions.

***Index Terms*—OMNeT++, DNS, mDNS**

I. Introduction

The domain name system specified by RFC 1035 [1] is used to name and share resources, making it a crucial part of the world wide web as we know it. Although DNS in general is a well-researched topic, its extensible nature continuously provides grounds for new research. With a simulation model, behavior and performance of DNS and extensions to DNS can be evaluated more rapidly without the need of using real systems. As an example, it may be easier to extend the simulation model by new caching strategies and evaluate these strategies by predefined measurements and behavioral studies than to integrate the strategies into an existing system and test it using real clients. It is also easier to instrument the simulation to capture statistical information, than in a real world system. We use the proposed models, e.g. to evaluate our *Stateless DNS* technique [2], which utilizes caching behaviors of DNS servers to answer queries without itself holding state.

We also present a model for multicast DNS as specified in RFC 6762 [3], which is widely used in local networks for which a dedicated DNS zone is not defined. Many of the specific implementation details are based on the implementation of Avahi [4], an open-source implementation that facilitates multicast DNS service discovery (mDNS/DNS-SD).

The privacy extension for mDNS/DNS-SD developed in [5], [6], [7] is one of the deciding factors for which these models have been developed. Most larger networks, such as campus networks, deactivate multicast by default, making it hard to scale experiments to larger scopes. Using a simulation model, this limitation can be eliminated to some extent, leaving the experiment limited only by available resources.

Our models allow researchers to design and evaluate both projects more rapidly using DNS, mDNS and extensions thereof. In this paper we provide an overview of the models' architectures and components to help researchers utilizing them. All models proposed in this paper are available as part of an open-source project hosted on GitHub[1].

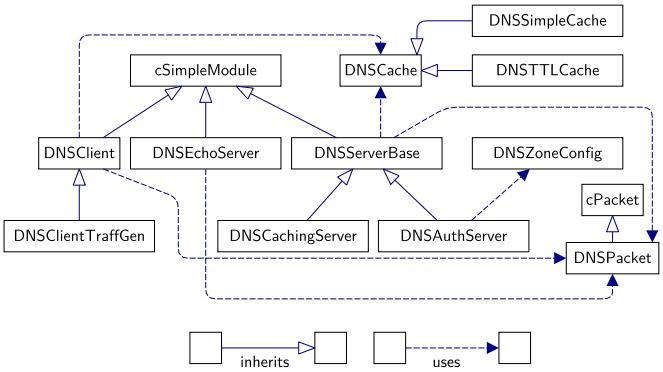


Fig. 1. Interaction diagram of the simple and compound modules implemented in the DNS model.

II. DNS Model

We implemented multiple components, each considering different aspects of the DNS architecture, including DNS servers, clients, and the functionalities needed to resolve and send queries. Figure 1 provides an overview of the interaction between the modules used within the DNS model. For comprehensibility the diagram does not include the relationships with lower level functions and structures needed for the implementation. As shown, the model consists of the following components:

`DNSServerBase`: This class provides the basic capabilities a DNS server should support. For instance, iterative querying is done within the server base, as well as caching of records. To achieve this, the server can use the generic caching interface provided by `DNSCache` and can utilize different caching strategies depending on the implementation.

`DNSAuthServer`: This module extends the basic module `DNSServerBase` and adds the functionality of an authoritative DNS server, meaning `DNSAuthServer` is authoritative for a specific zone and answers accordingly. The initialization of a zone configuration is done using the `DNSZoneConfig` class. Not all configurations that are

[1]Published at https://github.com/saenridanra/inet-dns-extension/

Example Configuration

```
$TTL 86400 ; 24 hours, $TTL used for all RRs
ORIGIN uni-konstanz.de.
@ IN SOA pan.rz.uni-konstanz.de.
         hostmaster.uni-konstanz.de. (
          2003080800 ; sn = serial number
          172800     ; ref = refresh = 2d
          900        ; ret = update retry = 15m
          1209600    ; ex = expiry = 2w
          3600       ; nx = nxdomain ttl = 1h
         )
  IN  NS pan.rz.uni-konstanz.de.    ; in the domain
  IN  NS uranos.rz.uni-konstanz.de. ; slave
  IN  MX imap.uni-konstanz.de.      ; external mail
  IN  A  134.34.240.80              ; ip of origin
; server host definitions
pan.rz       IN A      134.34.3.3    ; this server
uranos.rz    IN A      134.34.3.2    ; the slave server
imap         IN A      134.34.240.42 ; mail server imap
www          IN CNAME  proxy-neu.rz  ; test on
proxy-neu.rz IN A      134.34.240.80 ;
```

Fig. 2. Example zone configuration based on BIND[2] syntax.

typically possible with real world zone configurations, are currently implemented. Figure 2 shows a (working) sample configuration, based on the syntax used in BIND[2].

DNSCachingServer: This implementation can answer recursive queries by asking iteratively and performing cache lookups.

DNSEchoServer: The echo server is implemented according to [2] and provides the *echo domain* `.00.` and the *CCA* method `.cca.`.

DNSClient: Using this module, DNS servers can be queried. For this purpose, the module provides a function `resolve`, which takes multiple arguments needed for the query, as well as callback handles that are called when the query has been performed. This enables modules, which using this implementation to perform some operation on the received data.

DNSClientTraffGen: For simple simulation purposes, this module can be used to perform basic queries. Therefore, it needs to read the desired queries from a file, the location of which can be specified as a parameter within the NED description of this module. It then randomly chooses from the queries and periodically sends them to the DNS servers.

DNSCache: This class represents an interface for caching DNS records. Currently we have two implementations, `DNSSimpleCache` and `DNSTTLCache`, the former evicting records from the cache randomly and the latter based on the records' lifetime.

DNSPacket: The `DNSPacket` extends the basic packet class `cPacket` and adds four lists of resource records, i.e. for the question, answer, authority and additional sections each. Furthermore, it adds the options of a DNS packet and the ID as proposed in [1].

This list is not comprehensive, since it does not include all parts that are crucial to this model's implementation. For a detailed description of all modules, classes and functions, see our official documentation[3].

[2]https://www.isc.org/downloads/bind/

[3]http://saenridanra.github.io/inet-dns-extension/doc/neddoc/index.html

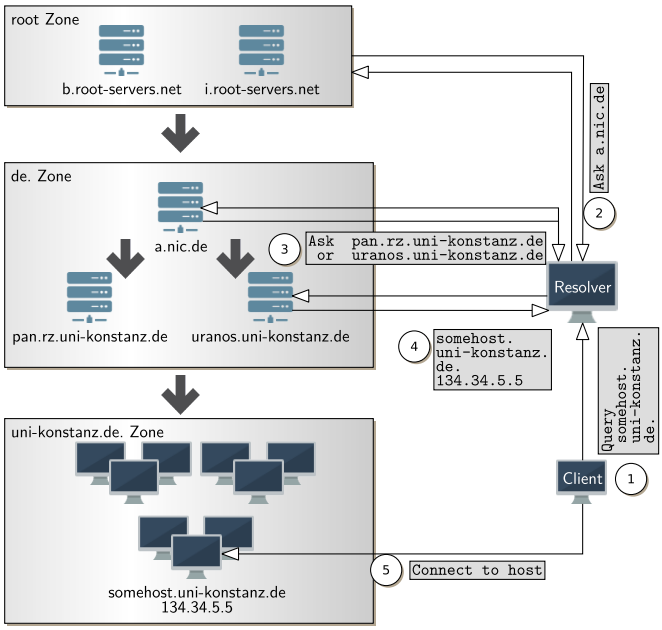


Fig. 3. This Figure shows a subset of an example network provided in the project source code, as well as the information flow in a live example.

Capabilities: At a high level the model's capabilities include the modeling of real world networks using DNS to resolve names, i.e. it supports the creation of hierarchical structures and is able to resolve names recursively and iteratively. To this end, a set of root servers must be defined, which are contacted by default if a resource's authoritative nameserver is unknown. Figure 3 shows an example DNS network and a client performing a query asking for `somehost.uni-konstanz.de`. Name compression is only considered when calculating the size of a packet, but not implemented for packets, since data structures are passed within the packet data structure and not as a payload consisting of only bytedata. However, following the example of the INET framework, we implemented a serializer for `DNSPacket`s that currently serializes `A`, `AAAA`, `NS`, `PTR`, `SRV`, `CNAME`, and `TXT` records properly and performs name compression where it is permitted.

Implementation challenges: OMNeT++ is well-suited for simulating the DNS protocol. The main difficulty resides in configuring a representative DNS network, and even more importantly, dynamically generating such networks. Another difficulty was analyzing and mapping existing rules of resolving and caching queries, since many of those rules are implementation-specific and not defined in the RFC. Providing extensibility and access in a generic way, as well as integrating the model within the INET framework to ease usage for researchers, were important concerns as well.

Limitations: Some functionalities have not been implemented, since they are not directly needed for the evaluations the models have been developed for:

- A DNS network has to be modeled manually. This

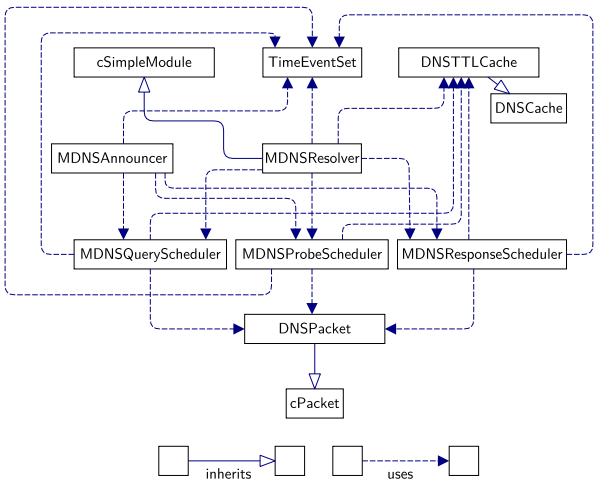


Fig. 4. Interaction diagram of the components implemented and used in the mDNS model.

includes defining the zone configurations for DNS servers and providing IP addresses accordingly. It would be preferable to assign IP addresses for a known host within a zone after automatic configuration using the `IPv4NetworkConfigurator`.

- Bailiwick rules as described in [8] are currently not implemented.
- Currently the DNS servers only reply properly to `A`, `AAAA`, `NS`, `MX`, `CNAME` and `ANY` queries. Other operations can be easily implemented, as placeholders are provided at corresponding positions.
- There is no support for dynamic zone updates of DNS servers.
- Extensions such as DNSSec [9] are currently not implemented.

III. mDNS Model

mDNS [3] provides local name resolution functionality. It is widely used in combination with DNS-SD [10] to provide zero configuration service discovery in local networks. The model proposed in this paper supports both mDNS and DNS-SD, one of the major goals being to evaluate the performance of mDNS/DNS-SD in networks in combination with the privacy extension and *Stateless DNS* [2]. Figure 4 provides an overview of how the different components in the model interact with each other.

As in Figure 1, only the most important components and their relations are shown, which are:

`MDNSResolver`: A simple module that essentially uses schedulers to query, probe and respond. It schedules *self-messages* according to the next event due in the set of time events and performs callbacks on elapsed events. It also handles the initialization of configured services, pairing data and private services.

`MDNSAnnouncer`: This class announces configured services to the network according to [3]. Hence, in a first step the services are probed for existence. If no conflict occurs, the service is announced to the network by sending unsolicited responses as defined in [3].

`MDNSProbeScheduler`: The probe scheduler probes the network with services that are to be announced, so that conflicts can be avoided. When a probe is posted and is not marked for immediate transmission, it is first put into a list and after a maximum of 250 ms, it is sent out along with other probes that need to be sent. Therefore, less packets are sent into the network. This timeframe also gives other devices the opportunity to respond on conflicting probes, that can be taken out of the schedule.

`MDNSQueryScheduler`: The query scheduler maintains queries that are to be sent out and additionally performs *duplicate question suppression* as defined in [3]. Additionally, *known answer suppression* is performed to further reduce the amount of traffic that is sent over the network.

`MDNSResponseScheduler`: In addition to maintaining response schedules, this class also performs *duplicate answer suppression* as defined in [3].

`TimeEventSet`: This class wraps a standard library container, more specifically an ordered set. The order is given by a `TimeEventComparator` that compares elements based on their expiry time. Since it is ordered, the head element of the set is always the next event due and since it is a set, events can be easily deleted and inserted, which would be more difficult with a priority queue. The time event set is used to maintain schedules and only set a single *self-message* based on the next due event, thereby improving the efficiency of the simulation. A time event is linked to a callback, so that a specific operation, such as checking if a probe can be sent out, can be performed when the event is due.

A component not shown in Figure 4, which is nevertheless crucial for evaluation, is the `MDNSNetworkConfigurator` module; it allows performing larger experiments without the need to configure resolvers manually. A network including this module is the *dynamic_mdns_network* example which is provided in the project source code. It can be configured to use a dynamic number of hosts, the number of services they use and different privacy related parameters that will be discussed in the context of the privacy extension.

Capabilites: At this time the model includes most functionalities described in [3]. In addition, it facilitates performing large experiments using dynamic parametrization. The startup procedure and announcement of static services is fully implemented.

Implementation challenges: Due to the restrictions set by [3] on how and when to send queries, probes and responses, scheduling, as it is done within the *Avahi Daemon*, is preferable. A simple solution would be to periodically set a *self-message* and check if an event is due. To improve efficiency, only one *self-message* is used based on the earliest event due and rescheduled in case of a new event that is due earlier. Another challenge is how to dynamically create and parameterize such networks, since it is hard to determine how

many devices in a network use mDNS and DNS-SD and how many services are used. By varying parameters like the number of mDNS/DNS-SD hosts and the number of services they can use different scenarios can be evaluated.

Limitations: There are some limitations and some functionalities currently not implemented:

- Shared resource records are not handled in separate data structures, as it is done in Avahi.
- Dynamic traffic generation is not implemented, meaning that resolvers will announce static services and periodically send unsolicited responses to reannounce the services, but will not query for services dynamically.
- Placeholders for internal module messages, initiating resolvers to query for services, are in place, but not yet implemented. These could be used by a compound module including a traffic generator to query for services and therefore simulate a more realistic network.
- Similar to the DNS model, not all operations defined by DNS and later extensions are supported for the mDNS model, for instance a query using the `AXFR` type.

IV. Privacy Extension

The idea of the privacy extension is to reduce the amount of information published with mDNS-SD and to reduce the amount of traffic transmitted by mDNS-SD. We only provide a brief overview of the privacy extension, since a detailed description can be found in [5], [6], [7].

To prevent private information being sent via multicast, the information is sent via a separate socket directly to trusted devices. The network parameters of this socket are offered and requested using a special meta-service that can distribute the desired information using alternative methods, e.g. our *Stateless DNS* technique [2]. Stateless DNS enables the use of mDNS-SD in networks where multicast is disabled because all information can directly be sent using unicast. While adding these features, the privacy extension is still fully backwards compatible. The presented models have been developed in order to measure the performance of mDNS-SD with and without the privacy extension, as well as using *Stateless DNS*. Using the `MDNSNetworkConfigurator` parameters can be varied to evaluate different scenarios. Parameters that can currently be modified include: number of resolvers, number of private resolvers, minimum and maximum number of friends and services. This already enables the simulation of large networks with mDNS-SD with a large number of resolvers, which could be used as an example of a network in an airport in which mDNS-SD is enabled.

Preliminary simulations have shown that our privacy extension significantly reduces multicast traffic and reduces network load by more than 50%. A sample of the results is shown in Figure 5.

V. Ongoing and Future Work

As part of our ongoing work, we want to add different traffic sources to the networks and evaluate how multicast in general and more specifically mDNS-SD affects the network performance of wireless networks. Additionally other concepts of reducing traffic generated by mDNS-SD are to be evaluated.

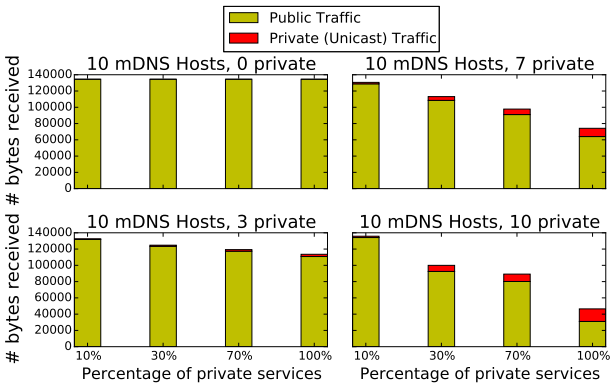


Fig. 5. Each subfigure shows a setup with ten mDNS hosts. The number of private hosts differs for each subfigure. The y-axis shows the number of bytes received. The x-axis shows the ratio of private services. With increasing privacy the traffic is reduced significantly.

Future work using the models presented in this paper may include:

- Dynamic generation of DNS networks.
- An implementation and evaluation of DNSSec [9].
- The implementation of host update protocols for DNS.
- Implementation and analysis of DNS caching (e.g. Bailiwick [8] rules) behavior to promote better caching rules and fine tune them towards performance and security.
- The evaluation of other experimental protocol extensions to DNS/mDNS without the need to test in the real world.
- Better integration of the models into the INET framework for ease of use.

References

[1] P. Mockapetris, "Domain names - implementation and specification," November 1987, RFC 1035.
[2] D. Kaiser, M. Fratz, M. Waldvogel, and V. Dietrich, "Stateless DNS," University of Konstanz, Tech. Rep. KN-2014-DiSy-004, Dec 2014.
[3] S. Cheshire and M. Krochmal, "Multicast DNS," February 2013, RFC 6762.
[4] "Avahi," http://avahi.org, Internet Resource, last visited on May 24th, 2015.
[5] D. Kaiser and M. Waldvogel, "Adding privacy to multicast DNS service discovery," in *Proceedings of IEEE TrustCom 2014 (IEEE EFINS 2014 Workshop)*, 2014.
[6] ——, "Efficient privacy preserving multicast DNS service discovery," in *Workshop on Privacy-Preserving Cyberspace Safety and Security (IEEE CSS 2014)*, 2014.
[7] D. Kaiser, A. Rain, M. Waldvogel, and H. Strittmatter, "A multicast-avoiding privacy extension for the Avahi zeroconf daemon," *Netsys 2015*, 2015.
[8] S. Son and V. Shmatikov, "The hitchhiker's guide to DNS cache poisoning," in *Security and Privacy in Communication Networks*. Springer, 2010, pp. 466–483.
[9] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS security introduction and requirements," March 2005, RFC 4033.
[10] S. Cheshire and M. Krochmal, "DNS-based service discovery," February 2013, RFC 6763.

Integration of RTMFP in the OMNeT++ Simulation Environment

Felix Weinrank
Münster University of Applied Sciences
Dept. of Electrical Engineering
and Computer Science
Bismarckstrasse 11
D-48565 Steinfurt, Germany
weinrank@fh-muenster.de

Michael Tüxen
Münster University of Applied Sciences
Dept. of Electrical Engineering
and Computer Science
Bismarckstrasse 11
D-48565 Steinfurt, Germany
tuexen@fh-muenster.de

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

***Abstract*—This paper introduces the new Real-Time Media Flow Protocol (RTMFP) simulation model for the INET framework for the OMNeT++ simulation environment. RTMFP is a message orientated protocol with a focus on real time peer-to-peer communication. After Adobe Inc. released the specifications, we were able to implement the protocol in INET and compare its performance to the similar Stream Control Transmission Protocol (SCTP) with a focus on congestion control and flow control mechanisms.**

I. Introduction

Real time audio and video communication has become a more and more important topic over time. Especially the increasing market penetration of mobile devices like smartphones and tablets presents new challenges for IP based communication protocols concerning address changes due to network handover, encryption and low latency. Protocols like the Real-Time Media Flow Protocol (RTMFP) [1] and WebRTC [2] have been developed to face these requirements. As Adobe published the specifications for RTMFP it was possible to implement an RTMFP model in the OMNeT++ / INET suite.

This paper will introduce RTMFP, its integration into the INET framework, the model validation and a comparison with the similar Stream Control Transmission Protocol (SCTP) [4].

Section II of this paper will give an overview of RTMFP itself, followed by the description of the protocol integration in the INET framework. Section IV will cover the validation of the new model followed by a comparison of RTMFP and SCTP with regard to performance and fairness.

II. The Real-Time Media Flow Protocol (RTMFP)

The Real-Time Media Flow Protocol (RTMFP), specified in RFC 7016 [1], is a peer-to-peer protocol for real time communication over IP based networks and has been developed by Adobe Systems Inc.

RTMFP is widely deployed on several platforms due to its integration in the Adobe Flash Player and Adobe Air, where it is implemented since 2008, whereas SCTP is an essential part of WebRTC where it is used for the WebRTC Data Channels [5]. The RFC 7016 did not specify specific encryption methods, these have been supplemented by Adobe in the RFC 7425 [3].

Its use cases are audio and video calls with low latency as well as bulk data transfers. In the beginning, the specifications of RTMFP were closed source but after several drafts, Adobe published the final RFC 7016 in November 2013. The message oriented, UDP based protocol contains features for NAT traversal and is encrypted by default - only address and session information remain unencrypted.

A. Sessions, Flows and Messages

A connection between two RTMFP peers is called session. Each session is bidirectional, congestion controlled and identified by its unique session identifier. In order to meet the demands of mobile environments, RTMFP supports an IP address change of a peer and connection re-establishment without interruption. The usage of a unique session identifier allows an address change of one peer without re-establishing the connection - the sender just continues sending packets and the receiver will recognize the peer address change and use the new address. To resist denial-of-service attacks, a four-way-handshake is used.

Sessions can contain zero or more unidirectional, message oriented flows for data transmission which are identified by a unique flow identifier. Flows are multiplexed, prioritized, allow in- or out-of-order delivery with variable reliability, and every flow has an independent flow control.

Each RTMFP message consists of a generic header and one or more chunks. The bundling of chunks reduces the protocol overhead especially if many small fragments are transmitted.

B. Congestion- and Flow-Control

Since RTMFP is UDP based, it has to implement its own congestion control and flow control. The built-in loss based and TCP-friendly congestion control manages an independent congestion window for each session and allows the prioritization of time critical data like audio or video over bulk data. When sending real time data on one session, the congestion control informs other local sessions and its peer about the time critical transfer. As a consequence, the other congestion

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

controls should switch to a less aggressive mode to prioritize real time data.

The congestion control is driven by acknowledgements and packet loss indications, sent by the receiver, and by timeouts. When the receiver acknowledges received packets to the sender, the sender increases its congestion window. If the sender detects packet loss due to loss indications by the receiver or timeouts, it will reduce the congestion window in relation to its operation mode. When the session operates in real time mode, the congestion control increases the window in a more aggressive way than in normal operation mode. The same procedure is applied for reacting on packet loss where the congestion control will reduce the congestion window in a less aggressive way in real time mode than in normal mode.

Every flow has its own flow control to manage the receiver buffer. In every acknowledgment chunk the receiver announces the available receive buffer size to the sender.

C. RTMFP and SCTP

RTMFP is quite similar to SCTP in many aspects. Both protocols use a four-way-handshake, support bundling and fragmentation and have the concept of streams/associations and flows/streams. Also like RTMFP, SCTP has an independent congestion control per association. In contrast to RTMFP, SCTP does not have built-in encryption but this can be achieved by the use of Datagram Transport Layer Security (DTLS) [6] like in the WebRTC stack where SCTP packets are encapsulated by DTLS.

III. The Simulation Model

The RTMFP simulation model for the INET framework consists of two simple modules called RTMFP-Layer and RTMFP-App, together they build the RTMFP-Compound-Module. An RTMFP-Compound-Module consists of one RTMFP-Layer instance and one or more RTMFP-Apps and is connected to the INET Standardhost by a UDP gate - see Figure 1. As we were particularly interested in the transmission performance of the protocol, we focused on the connection establishment, chunk bundling, flow control and the congestion control. These features are completely covered by the simulation model, wheras features like NAT traversal or encryption are not implemented yet.

A. RTMFP-App

The RTMFP-App module simulates an application which makes use of RTMFP. Each RTMFP-App represents an independent endpoint that can receive data and may also act as a sender to initiate a session to another endpoint with one or more independent flows. Each flow is configured separately with respect to its packet rate, send interval, maximum number of packets and its absolute prioritization. The RTMFP-App records statistical data for every flow like transfer rate, number of received and sent messages and runtime.

B. RTMFP-Layer

The RTMFP-Layer represents the operation of RTMFP. All RTMFP specific features like connection establishment, session management, flow control and congestion control have been implemented in this module. The strict separation in RTMFP-Applications and RTMFP-Layer allows us to quickly create new applications for testing scenarios while keeping them as simple as possible.

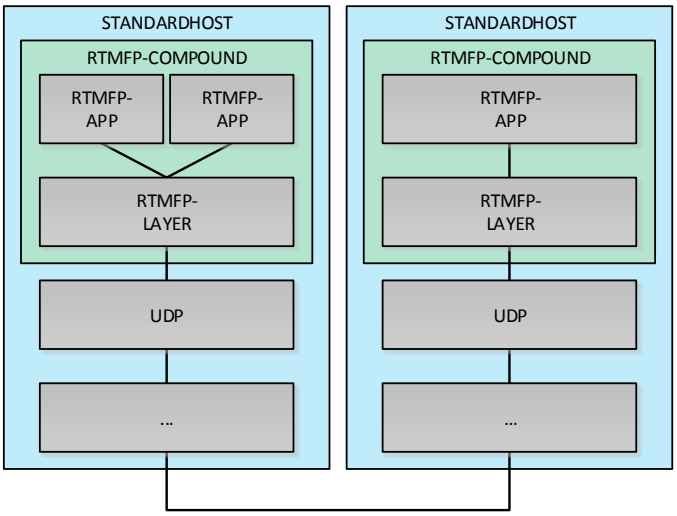


Fig. 1. RTMFP-Compound-Module architecture

C. Model Parameters

To cover a wide range of peer-to-peer communication scenarios from single file transfer to more complex scenarios like audio and video conferences with parallel file transfer, it is necessary to make the configuration as customizable as possible. We fulfilled this requirement by a flexible architecture where the user can add any number of RTMFP-Apps with any number of flows by declaring them in the configuration file.

Figure 2 shows a sample configuration for an RTMFP-Compound-Module. `numRtmfpApps` defines the number of RTMFP-Apps attached to the RTMFP-Layer.

The RTMFP-Layer can be configured by setting its UDP port (`localPort`), maximum packet size (`maxSegmentSize`), the initial congestion window (`ccCwndInit`) and the size of the receive buffer for incoming flows (`rcvBufferSize`). The initial congestion window is set by `ccCwndInit`.

Every RTMFP-App has its own subset of settings: `flowsOutgoing` sets the number of outgoing flows. All outgoing flows are configured by a list of parameters, separated by spaces. The size of every outgoing packet (`flowPacketSize`), the interval between every send call (`flowSendInterval`) and the amount of outgoing packets (`flowNumPackets`) can be configured as fixed values or by using any OMNeT++ built-in distribution like `exponential` or `uniform`. If a built-in distribution is chosen it will be evaluated by every call of the RTMFP-App send function. `readDelay` is the timespan the application will wait to request data from the layer after being notified about arrived data, this is useful to fill the receive buffer.

```
# HOST 1
**.udpApp[0].numRtmfpApps = 2
**.udpApp[0].layer.localPort = 4711
**.udpApp[0].layer.maxSegmentSize = 1472byte
**.udpApp[0].layer.rcvBufferSize = 65536byte
**.udpApp[0].layer.ccCwndInit   = 4380byte

# APP 0
**.udpApp[0].app[0].localEpd = 4712
**.udpApp[0].app[0].remoteAddress = "host2"
**.udpApp[0].app[0].remotePort = 2013
**.udpApp[0].app[0].remoteEpd = 2014

**.udpApp[0].app[0].flowsOutgoing = 2
**.udpApp[0].app[0].flowPacketSize = "140byte 140byte"
**.udpApp[0].app[0].flowSendInterval = "1000us 1000us"
**.udpApp[0].app[0].flowNumPackets = "500000 500000"
**.udpApp[0].app[0].flowTimeCritical = "1 1"
**.udpApp[0].app[0].flowId = "19 88"
**.udpApp[0].app[0].maxRuntime = 1800s
**.udpApp[0].app[0].readDelay = 0ms

# APP 1
...
```

Fig. 2. Simulation parameters for an RTMFP-Layer and an RTMFP-App with two outgoing flows

D. Model Operation

When starting the simulation, the RTMFP-Apps register themselves at the layer with their unique Endpoint-Discriminator (EPD) and, if configured to do so, they trigger the RTMFP-Layer to establish a session with the given peer, also identified by its EPD, and one or more address candidates. When the RTMFP-Layer has successfully established a connection, it informs the RTMFP-App about the new session and the RTMFP-App starts sending data to the peer. The RTMFP-Layer and RTMFP-Apps use a custom set of cMessages to communicate. When an RTMFP-App is configured to start a data transfer to a peer, it starts sending data messages to the RTMFP-Layer which are queued in the given send flow.

If a single RTMFP-App message exceeds the Maximum Transmission Unit (MTU) it will be fragmented by the sending RTMFP-Layer into matching chunks and reassembled by the receiving RTMFP-Layer. Despite other attributes, every queued message gets an increasing sequence number, an indication if it is a complete message or a fragment and a priority marker, i.e. time critical or not.

If the flow control of the specific flow and the congestion control allow sending data, the RTMFP-Layer tries to bundle as many chunks as possible into one RTMFP packet - preserving the MTU.

When receiving a data chunk from a peer, the RTMFP-Layer will put the payload into the receiver queue of the corresponding receive flow and send a notification to the associated local RTMFP-App. The receiver should acknowledge every second received data message by an acknowledgment chunk, unless a packet loss has been detected.

When receiving an acknowledgment chunk, the sender deletes successfully transmitted messages from its queue and adapts the flow control and the congestion window. A message that is reported missing three times should be handled as lost and be retransmitted by the sender. This will all be done by the RTMFP-Layer, the RTMFP-App will not be notified. The acknowledgment chunks have a direct impact on the congestion control, every time the receiver reports a successful transmission without a loss report, the congestion control increases the congestion window, depending on its operation mode. In the case that a packet loss is detected by a loss report, the congestion control decreases the congestion window.

After all messages have been sent or the maximum simulation runtime is reached, all RTMFP-Layer and RTMFP-Apps record their statistics and shut down.

IV. Model Validation

To validate the RTMFP protocol implementation, we examined a set of predictable simulation scenarios to test the implemented features - especially flow control, congestion control and bundling were in our focus.

Due to the lack of comparable RTMFP models and the fact that RTMFP is only available as a commercial product via Adobe Flash or Adobe Air, it was not possible to cross-validate our model with real implementations.

To achieve a more realistic testing scenario and get some randomness, we used a bottleneck link scenario where the RTMFP traffic competed with a random UDP sender on the bottleneck which sent a small amount of traffic - see Figure 3.

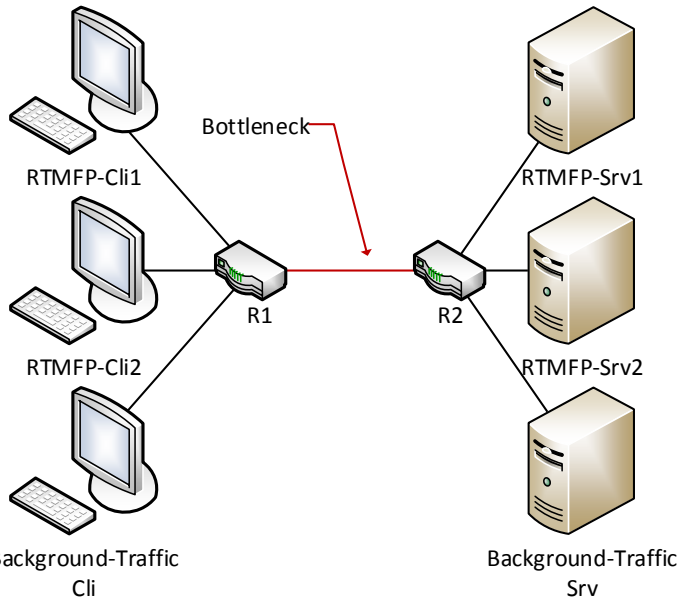


Fig. 3. Bottleneck scenario with two RTMFP peers and background traffic

- The **flow control** has been validated by using the bandwidth delay product (BDP) which sets the bandwidth in relation to the link delay. We used a bottleneck scenario with a variable delay from 0 to 100 milliseconds, measured the bandwidth and compared the results with our theoretical calculations.
- Two or more competing RTMFP-Clients share a bottleneck for **congestion control** testing. In the first scenario all clients start sending simultaneously. See Figure 4. In the second scenario the clients start sending successively. The congestion control mechanisms should ensure a fair distribution of the available bandwidth.
- We varied the size of the applications data messages and recorded the bandwidth to prove the **bundling** mechanism. As every data chunk has a given header size, small data chunks have a low payload to header size ratio and this will result in a lower payload transfer rate.

The testing scenarios reached our expectations.

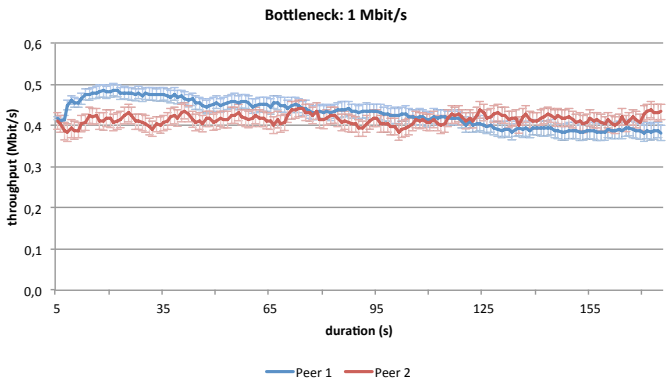


Fig. 4. Bottleneck scenario with two RTMFP peers and random traffic

V. Comparing RTMFP to SCTP

After we had tested the RTMFP model in relevant aspects, we were able to compare RTMFP with the OMNeT++ / INET SCTP model [7]. One of our primary interests was the comparison of RTMFP and SCTP in relation to fairness and performance under different conditions like bottleneck-scenarios, lossy and delayed links.

Figure 5 shows the impact of packet loss on the transfer rate of both protocols. Whereas both protocols show a similar bandwidth performance in a scenario without loss, SCTP delivers nearly double the amount of data under packet loss conditions.

In direct comparison with a bottleneck scenario, RTMFP and SCTP share the available bandwidth nearly fair under most scenarios. When RTMFP operates in the normal operation mode - without real time traffic - the congestion control is less aggressive than the SCTP congestion control, which meets our expectations.

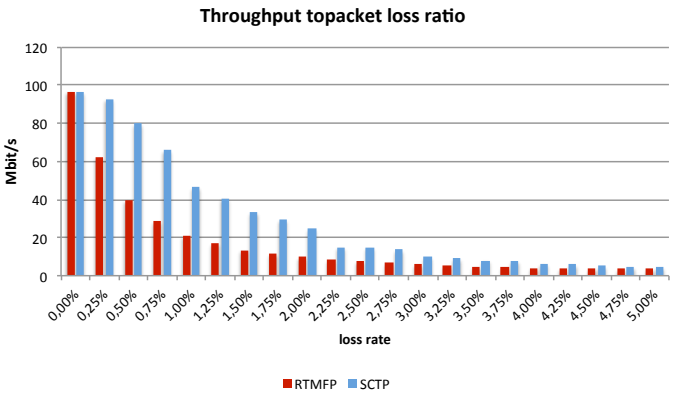


Fig. 5. Bottleneck scenario with two RTMFP peers and random traffic

VI. Conclusion and Outlook

The implemented RTMFP simulation model covers nearly all specified features of the protocol which are relevant for connection establishment and data transfer, i.e. four way handshake, congestion control, flow control, chunk bundling and support for multiple sessions and flows. This allows a detailed analysis of the built-in congestion control and flow control mechanisms.

With the new model we were able to compare RTMFP to the quite similar SCTP and measure the performance of both protocols under different conditions. We are currently working on a connecting module to use INET's Netperfmeter module with RTMFP. This will allow us to use the Netperfmeter as a traffic generator for RTMFP and SCTP. We are planning to contribute the RTMFP model to the INET suite.

References

[1] M. Thornburgh, "Adobe's Secure Real-Time Media Flow Protocol", RFC 7016, November 2013

[2] S. Loreto, S.P. Romano, "Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts", Internet Computing, IEEE (Volume:16 , Issue: 5) Page(s): 68 - 73 Sept.-Oct. 2012

[3] M. Thornburgh, "Adobe's RTMFP Profile for Flash Communication", RFC 7425, December 2014

[4] R. Stewart, Ed., "Stream Control Transmission Protocol", RFC 4960, September 2007

[5] R. Jesup, S. Loretto, M. Tüxen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-13, January 4, 2015

[6] M. Tüxen, R. Stewart, R. Jesup, S. Loreto, "DTLS Encapsulation of SCTP Packets", draft-ietf-tsvwg-sctp-dtls-encaps-09, January 24, 2015

[7] I. Rüngeler, M. Tüxen, E. Rathgeb, "Integration of SCTP in the OMNeT++ Simulation Environment", Proc. of the 1st international conference on Simulation tools and techniques for communications, networks and systems workshops, March 3, 2008

ptp++: A Precision Time Protocol Simulation Model for OMNeT++ / INET

Martin Lévesque and David Tipper
School of Information Sciences
University of Pittsburgh
Pittsburgh, PA, USA

***Abstract*—Precise time synchronization is expected to play a key role in emerging distributed and real-time applications such as the smart grid and Internet of Things (IoT) based applications. The Precision Time Protocol (PTP) is currently viewed as one of the main synchronization solutions over a packet-switched network, which supports microsecond synchronization accuracy. In this paper, we present a PTP simulation model for OMNeT++ INET, which allows to investigate the synchronization accuracy under different network configurations and conditions. To show some illustrative simulation results using the developed module, we investigate on the network load fluctuations and their impacts on the PTP performance by considering a network with class-based quality-of-service (QoS) support. The simulation results show that the network load significantly affects the network delay symmetry, and investigate a new technique called class probing to improve the PTP accuracy and mitigate the load fluctuation effects.**

I. Introduction

Precise time synchronization is a key requirement in several packet based communication networks and real-time networked application domains, such as the automated industrial systems and smart power grid systems. For instance, the communication technologies such as the Long Term Evolution-Advanced (LTE-A) cellular networks require backhaul equipment base stations to provide time synchronization in order to synchronize transmissions over frequencies from adjacent base stations and interference coordination. Also, the emerging smart power grid systems, which are characterized by a two-way flow of energy and end-to-end communications, will also require tight time synchronization. In those systems, the communications are machine-to-machine in nature and require synchronized information in order to improve reliability and efficiency of power delivery.

A general solution to provide the synchronization functionality among networked devices requiring time alignment is to incorporate an atomic clock or a Global Positioning System (GPS) component in each device. However, equipping each device of these technologies would be extremely costly, especially for instance in sensor and actuator devices, which in many applications are cost and computationally constrained. To reduce these cost issues, network protocols have been developed to distribute time over packet-switched networks and synchronized distributed devices. The Network Time Protocol (NTP)[1] is a widely used Internet time synchronization protocol, which provides millisecond synchronization accuracy and is implemented at the application layer. However, millisecond accuracy is not sufficient for all applications requiring synchronization. To provide precise time synchronization, the Precision Time Protocol (PTP) was then proposed [1], which operation principle is similar to NTP, but provides new features to meet microsecond synchronization accuracy.

Given the emergence of distributed and real-time applications requiring tight time synchronization performance, there is a growing need to study time synchronization for these applications in a low-cost manner. PTP is currently one of the most investigated synchronization protocols, but there is still a lack of module compliant with the OMNeT++ INET framework to study PTP-based applications over wired and wireless networks. In [2], the authors developed a PTP simulation model over IEEE 802.11 networks. However, the PTP module was added using 802.11 modules specifically without following the OMNeT++ node structures, and the implementation is currently not available to the best of our knowledge. In our implementation, PTP can be used with any communications technology (e.g., Ethernet, 802.11, etc.), since PTP is an application layer protocol and uses the above network layers. In this paper, we present our developed PTP module for OMNeT++ INET, tested with the most recent version (INET 2.6 for OMNeT++ 4.4). To show some illustrative simulation results, we investigate the PTP performance under the presence of variable network loads with (and without) quality-of-service (QoS) support, and propose a new mechanism to improve the PTP accuracy.

The remainder of the paper is structured as follows. In Section II, we overview key background information on the Precision Time Protocol (PTP). In Section III, the PTP implementation model for OMNeT++ is next described. The simulation results are then provided in Section IV. Conclusions are finally drawn in Section V.

This work was supported by NSERC Postdoctoral Fellowship No. 453711-2014.

Corresponding author: Martin Lévesque, School of Information Sciences, University of Pittsburgh, Pittsburgh, PA 15260 (email: levesque@pitt.edu).

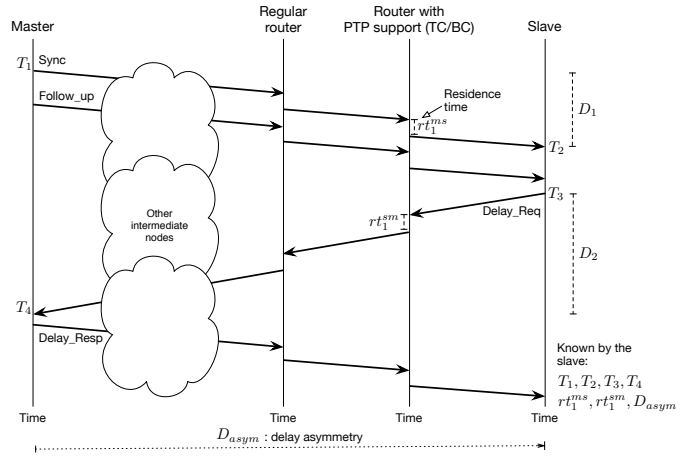


Fig. 1: The Precision Time Protocol (PTP) over a network with a subset of the routers having PTP support. (TC: Transparent clock, BC: Boundary clock)

II. Background

In this section, we describe some key PTP notions, which we investigate in this paper using our developed OMNeT++ module. The Precision Time Protocol (PTP) version 2, IEEE 1588 [1], enables precise synchronization of clocks over heterogeneous systems with accuracy in the microsecond to sub-microsecond range. PTP was proposed to meet tighter synchronization accuracy compared to the widely used NTP protocol, where NTP targets distributed applications to meet millisecond synchronization requirements. On the other hand, PTP was designed to reach precise synchronization requirements for industrial automation, power systems, and telecommunications applications. In a fashion similar to NTP, PTP operates at the application layer. Further, each PTP node contains one or many ports, and communicates with other PTP nodes in a given network by implementing a synchronization protocol described hereafter.

A. PTP Synchronization Protocol

The main procedures of the PTP synchronization protocol, depicted in Fig. 1, are listed as follow:

- The master node periodically transmits a *Sync* message to the slave nodes containing the sent time T_1.
- A *Follow_up* message which contains T_1 is optionally sent depending on the timestamp processing mechanism.
- When T_1 arrives at a given slave node, the received time T_2 is recorded.
- The slave node then sends a *Delay_req* message containing T_3 to the master clock.
- Finally, the master node records the reception time T_4 and sends it back to the slave node in a *Delay_resp* message.

[1]The RFCs and relevant information on NTP are available online: http://www.ntp.org/.

Both the NTP and PTP protocols follow a similar strategy to update the clocks. First, the offset time at a given slave node k is approximated by:

$$\Theta_k = \frac{D_1 - D_2}{2}, \tag{1}$$

where $D_1 = T_2 - T_1$ and $D_2 = T_4 - T_3$ correspond to the downstream (from the master node to slave node) and upstream (from a slave node to master node) delays, respectively. To correct its clock, a given slave node k adjusts its local time t_k to:

$$t_k \leftarrow t_k - \Theta_k. \tag{2}$$

It is worth noting that Eq. (1) approximates the offset precisely if the messages experience symmetrical delays, that is, if both D_1 and D_2 are close. The presence of asymmetrical delays can significantly degrade the synchronization accuracy.

B. PTP Delay and Offset Improvement Mechanisms

PTP introduces three main improvement mechanisms to mitigate the negative effects of asymmetric links on the synchronization accuracy:

- *Residence time at intermediate nodes:* The residence time at a given router corresponds to the time duration a PTP packet resides in a switching fabric from the input port to the output port. In Fig. 1, two residence times are recorded, rt_1^{ms} and rt_1^{sm}, which are measured at the router with PTP support.
- *Asymmetric delay parameter, D_{asym}:* If the asymmetric delay properties are known in a given network, an asymmetric delay parameter can be used [3], which corresponds to the *delayAsymmetry* field in IEEE 1588.
- *Peer-to-peer path correction:* Peer-to-peer transparent ports measure the link propagation delays. Such a mechanism helps at reducing asymmetry, at the expense of an increased cost.

All of these techniques can be integrated in Eq. (1), as described in [1].

C. Performance

One significant metric to quantify the PTP performance is the synchronization error, which is the average time deviation between the slaves and master clocks. The PTP performance can vary significantly depending on the network and conditions (traffic load, asymmetry, timestamping method, etc.). In testbeds following most of the PTP standard recommendations, a synchronization accuracy of approximately 50 ns was achieved [4], [5]. Further, a synchronization precision of 2 ns under ideal conditions was obtained by implementing all PTP features in hardware, including the timestamping function [6]. However, under the presence of multi-hop communications with non PTP routers, significantly worst synchronization accuracies of 450 μs were measured under the presence of asymmetrical delays [7]. The investigation of the PTP performance is convenient by using an event-driven simulator

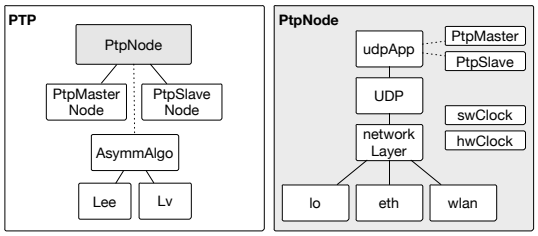


Fig. 2: Main components of the PTP simulation model.

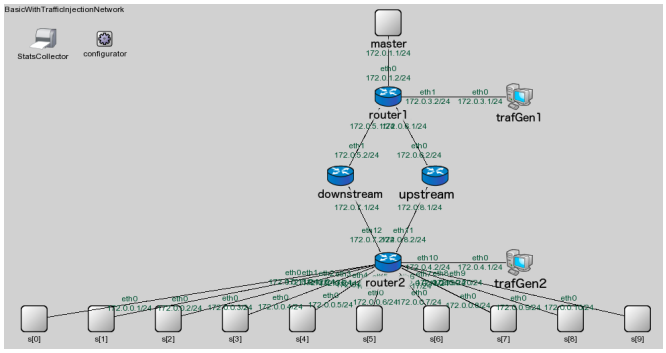


Fig. 3: Simulation scenario consisting of 10 slaves synchronizing to a master node over intermediate Ethernet routers.

such as OMNeT++, especially to evaluate new mechanisms, as we show shortly in Section IV.

III. Implementation Details

We next describe our PTP simulation model, ptp++[2], which architecture is depicted in Fig. 2. We adapted and extended the module developed in [8]. The main components of our model are described as follow:

- *PtpNode:* This class models a PTP node, which can be either a master or slave. As PTP operates at the application layer, both the master and slave applications are modeled as a User Datagram Protocol (UDP) application (*udpApp*). In order to be compliant with the OMNeT++ INET 2.6 networking framework, *PtpNode* follows a structure similar to *NodeBase*, thus implementing the overall UDP/Internet Protocol (IP) stack structure, including the application layer (*udpApp*), UDP, network layer, Medium Access Protocol (MAC), and physical layer.
- *Software and hardware clocks:* As the goal of PTP is to synchronize clocks, a hardware clock needs to be modeled at each node. The implemented hardware clock (*hwClock*) can take different drift distributions based on the model proposed by [8], [9]. The software clock (*swClock*) takes the hardware clock signals as input, and a processing delay can be added to model variable software impairments.
- *AsymmAlgo:* As we mentioned in the previous section, asymmetrical connections can significantly degrade the PTP accuracy. Recently, several mechanisms (e.g., Lee [10] and Lv et al. [11]) have been proposed to detect and mitigate the negative effects of the asymmetry on the PTP accuracy. These mechanisms periodically send extra control messages after the PTP execution. Thus, we developed the simulation model such that currently proposed and new asymmetry mitigation techniques can be evaluated.

Further, a *StatsCollector* module records the clock deviations in order to evaluate the accuracy performance under different network conditions. When a software clock time varies, the synchronization error is computed and added in the *StatsCollector*. In the current implementation, three statistic output files are generated at the end of a given simulation:

- *Summary text file:* The average synchronization error, standard deviation, minimum, and maximum values are appended in a general statistic file. If multiple configurations are executed, such a file can be used for plotting results under variable conditions.
- *Probability distribution function:* Using the recorded synchronization errors, probabilities are computed to represent the distribution of the synchronization errors.
- *Deviation vector*: An OMNeT++ vector of the synchronization errors is also recorded such that the errors can be visualized at the end of a given simulation.

Fig. 3 depicts a sample scenario consisting of 10 slave nodes synchronizing to a master node over three Ethernet routers. Depending on the load variations in the network, the PTP accuracy varies significantly, as we will show in the next section. Further, as recommended in the PTP standard, to improve accuracy, PTP messages must be processed as soon as they arrive in the router queues, where PTP messages are filtered with high quality-of-service (QoS) class priority. We implemented a class-based queue *PtpPrioritizedQueue* which processes PTP messages first by following a deep packet inspection procedure, which we investigate in the following results section.

IV. Simulation Results

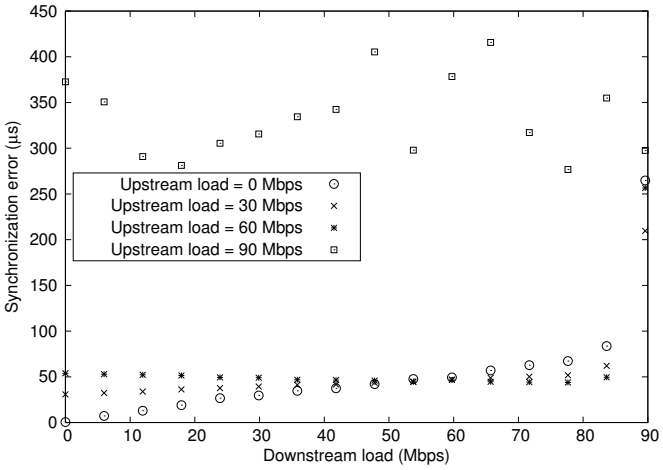
In this section, we investigate the network depicted in Fig. 3 in terms of PTP synchronization accuracy under variable network loads using our developed PTP simulation model. All communications interfaces are based on Ethernet with 100 Mbps full duplex capacity. The communications links are symmetric in terms of propagation and transmission delays. The asymmetry component we consider corresponds to the variable queuing delays experienced mainly in the router switching fabrics (e.g., nodes *router1*, *router2*, *downstream*, and *upstream*). In order to vary the network load, we have two traffic generators, *trafGen1* and *trafGen2*, which exchange messages between each other using the intermediate routers. Further, we configure the network routing tables such that the packets coming to *router1* and destinated to nodes *s[1..10]* and *trafGen2* are routed to the *downstream* and *router1*

[2]ptp++: https://github.com/martinlevesque/ptp-plusplus.

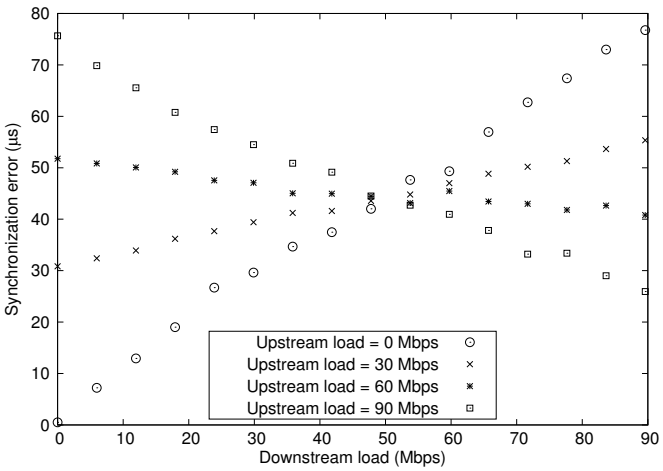
nodes. Similarly, the packets coming in *router2* and destinated upstream are routed to the *upstream* and *router1* nodes. Therefore, the downstream and upstream delays experienced by the PTP messages vary depending on the traffic load generated by the *trafGen1* and *trafGen2* nodes. Further, to simulate realistic and bursty traffic patterns, the traffic generators follow a Pareto distribution such that the average interarrival time between the generated packets equals $\mu = \frac{a \cdot b}{a-1}$, where $a = 1.5$ and b are the shape and scale of the Pareto distribution, respectively.

Fig. 4a) depicts the synchronization accuracy under variable upstream and downstream load conditions without using any QoS mechanism[3]. When the traffic load is close to 0 Mbps in the upstream and downstream directions, as expected, the synchronization error is below 10 μs. However, as the traffic increases, the synchronization error grows, especially when the network is under asymmetrical traffic load conditions (e.g., when the upstream load equals 90 Mbps with 0 Mbps downstream load). To improve these performance, we next configure prioritized queues in order to process PTP messages first, as discussed in the previous section. Fig. 4b) shows significant accuracy improvement while using priority queues, where the synchronization error varies between 0-80 μs compared to 0-400 μs while not using any QoS mechanism. As in the previous results (without using any QoS), when the traffic load is asymmetrical, the synchronization error increases. However, we observe that when priority queues are used at high loads, the synchronization error decreases compared at medium loads. This is due to the fact that at high load, the probability that a transmission of a non PTP packet occurs, while a PTP packet arrives at the same time, is higher.

We next propose a new mechanism called *class probing*, which consists of sending an extra non PTP message (and thus low priority) prior to sending a PTP message in order to increase the transmission probability of non PTP message in both directions, to improve asymmetry and consequently the synchronization accuracy. We show a comparison of using this technique vs. using the PTP standard in Fig. 5. We observe a significant improvement of sending an extra non PTP message prior to sending a PTP message, which will be investigated more extensively in future work.



(a) Without QoS.



(b) With QoS.

Fig. 4: Synchronization accuracy with variable upstream and downstream network loads.

V. Conclusions

In this paper, we described our PTP simulation model which extends the OMNeT++ / INET framework. Our implementation follows the PTP standard and allows to measure the protocol under different network configurations and conditions. We shown, using the implemented model, that the synchronization accuracy gets significantly affected by variable network loads, and using a prioritized QoS mechanism significantly improves the accuracy. The simulation model allows to investigate new PTP mechanisms under different settings at low-cost. The framework can also be used for time-aware applications, or other protocols requiring tight synchronization. For instance, some security protocols exchange timestamps to detect replay attacks [12]. Further, the emerging Internet-of-Things applications require to timestamp events, thus using a time synchronization protocol such as PTP improves the simulation realism with non perfect synchronized clocks.

[3]For each given upstream/downstream pair, we repeated the experiment 15 times with different random seed numbers.

References

[1] IEC Technical Committee 65 and IEEE Standards Association (IEEE-SA) Standards Board, "IEEE Standard for a precision clock synchronization protocol for networked measurement and control systems," *IEC 61588:2009(E) IEEE Std. 1588(E)-2008*, pp. C1–274, Feb 2009.

[2] Y. Huang, Y. Yang, T. Li, and X. Dai, "An Open Source Simulator for IEEE 1588 Time Synchronization over 802.11 Networks," in *Proc., IEEE European Modelling Symposium*, pp. 560–565, 2013.

[3] N. Simanic, R. Exel, P. Loschmidt, T. Bigler, N. Kerö, "Compensation of Asymmetrical Latency for Ethernet Clock Synchronization," in *Proc., IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, pp. 19–24, 2011.

[4] D. M. E. Ingram, P. Schaub, D. A. Campbell, and R. R. Taylor, "Evaluation of Precision Time Synchronisation Methods for Substation Applications," in *Proc., IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, pp. 1–6, 2012.

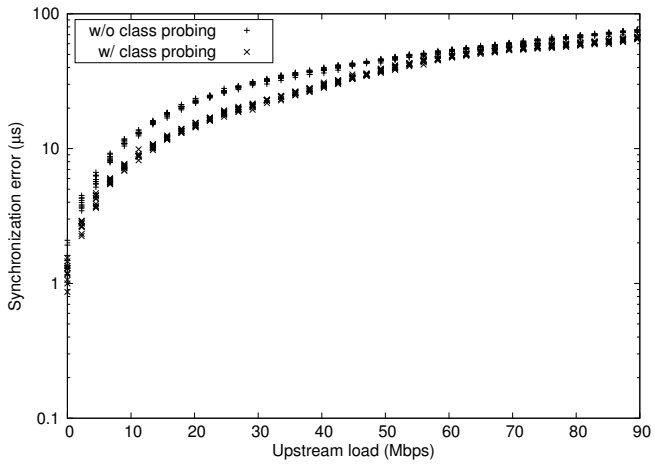


Fig. 5: Improvement of the synchronization accuracy by using a class probing mechanism.

[5] D. Latremouille, K. Harper, and R. Subrahmanyan, "An Architecture for Embedded IEEE 1588 Support," in *Proc., IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, pp. 128–133, 2007.

[6] D. Rosselot, "Simple, Accurate Time Synchronization in an Ethernet Physical Layer Device," in *Proc., IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, pp. 123–127, 2007.

[7] R. Zarick, M. Hagen, and R. Bartos, "The Impact of Network Latency on the Synchronization of Real-World IEEE 1588-2008 Devices," in *Proc., IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, pp. 135–140, 2010.

[8] J. Steinhauser, "*A PTP Implementation in OMNET++*, Treitlstr. 3/3/182-1, 1040 Vienna, Austria Technische Universität Wien, Institut für Technische Informatik," 2012.

[9] F. Ferrari, A. Meier, and L. Thiele, "Accurate Clock Models for Simulating Wireless Sensor Networks," in *Proc., 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools'10)*, pp. 21:1–21:4, 2010.

[10] S. Lee, "An Enhanced IEEE 1588 Time Synchronization Algorithm for Asymmetric Communication Link using Block Burst Transmission," *IEEE Communications Letters*, vol. 12, pp. 687–689, Sept. 2008.

[11] S. Lv, Y. Lu, and Y. Ji, "An Enhanced IEEE 1588 Time Synchronization for Asymmetric Communication Link in Packet Transport Network," *IEEE Communications Letters*, vol. 14, pp. 764–766, Aug. 2010.

[12] S. Uludag, K.-S. Lui, and K. Nahrstedt, "Secure and Scalable Data Collection With Time Minimization in the Smart Grid," *IEEE Transactions on Smart Grid (Early Access)*.

High Frequency Radio Network Simulation Using OMNeT++

Jeffery Weston
Eric Koski
RF Communications Division
Harris Corporation
Rochester, NY, USA

***Abstract*— Harris Corporation has an interest in making HF radios a suitable medium for wireless information networks using standard Internet protocols. Although HF radio links have many unique characteristics, HF wireless subnets can be subject to many of the same traffic flow characteristics and topologies as existing line-of-sight (LOS) radio networks, giving rise to similar issues (media access, connectivity, routing) which lend themselves to investigation through simulation. Accordingly, we have undertaken to develop efficient, high-fidelity simulations of various aspects of HF radio communications and networking using the OMNeT++ framework. Essential aspects of these simulations include HF channel models simulating relevant channel attributes such as Signal to Noise Ratio, multipath, and Doppler spread; a calibrated physical layer model reproducing the error statistics (including burst error distributions) of the MIL-STD-188-110B/C HF modem waveforms, both narrowband (3 kHz) and wideband (up to 24 kHz) on the simulated HF channels; a model of the NATO STANAG 5066 data link protocol; and integration of these models with the OMNeT++ network simulation framework and its INET library of Internet protocol models. This simulation is used to evaluate the impacts of different STANAG 5066 configuration settings on TCP network performance, and to evaluate strategies for optimizing throughput over HF links using TCP Performance Enhancing Proxy (PEP) techniques.**

Keywords—protocols, radio networks, HF channel characteric models, HF network simulation framework, HF radio network, Harris Corporation, OMNeT++ open-source network simulator, STANAG 5066 HF data link protocol, MIL-STD-188-110C, wideband HF waveforms, network simulation, radio, wireless.

I. Introduction

High Frequency (HF) radio has long been considered to have limited potential for data communications [1], due to highly variable propagation conditions, high error rates, long and variable latency, and data rates that (in narrowband channels) do not exceed and frequently cannot reach 9600 bits per second (BPS).

However, HF has several advantages that can offset the disadvantages. Foremost is the fact that HF radio signals can reliably propagate beyond line-of-sight (BLOS, also referred to as skywave), often around the world, with no dependence on a fixed network infrastructure.

BLOS communications over long distances can typically be achieved only within a 3 to 6 MHz wide band of the overall HF spectrum for any given time and station locations. Which band can be used depends on the time of day, season, and solar radiation characteristics that can be predicted *a priori* with only partial success. For this reason, most critical HF communications systems rely on near-real-time propagation sensing and frequency adaptation through the use of Automatic Link Establishment (ALE) techniques.

HF also exhibits interesting groundwave propagation mode which can result in successful propagation over distances somewhat longer than line-of-sight. HF groundwave propagation requires an entirely separate propagation model, and is the subject of distinct research and study.

II. Network Model Overview

It is useful to view a network model as comprising three parts [5]:

- Traffic model – the nature and behavior of the traffic source and sink, including arrival rate, "burstiness", Quality of Service (QoS) attributes, etc.
- Bearer model – the parts of the network that are actually used to transport the required traffic.
- Channel model – the medium across which the traffic is delivered.

We will discuss each of these sub-models in reverse order, since the unique needs of the channel model drive the other sub-model requirements. Refer to Figure 1 for a graphical view of the module layout and interconnections.

A. Channel Model

The physical factors affecting HF propagation are materially different than those affecting LOS radio propagation. The primary considerations in LOS radio propagation are free space loss (inverse-square distance loss),

ground reflection (as represented by two-ray propagation models), shadowing (by terrain or artificial features) and diffraction. Most of these are of secondary importance to HF skywave propagation, where the primary factors are ionospheric refraction and absorption spread due to simultaneous propagation paths with differential delay, multipath, and comparatively large Doppler spreads. HF groundwave propagation has its own set of unique considerations, including ground conductivity and permittivity, as well as shadowing and diffraction by terrain features.

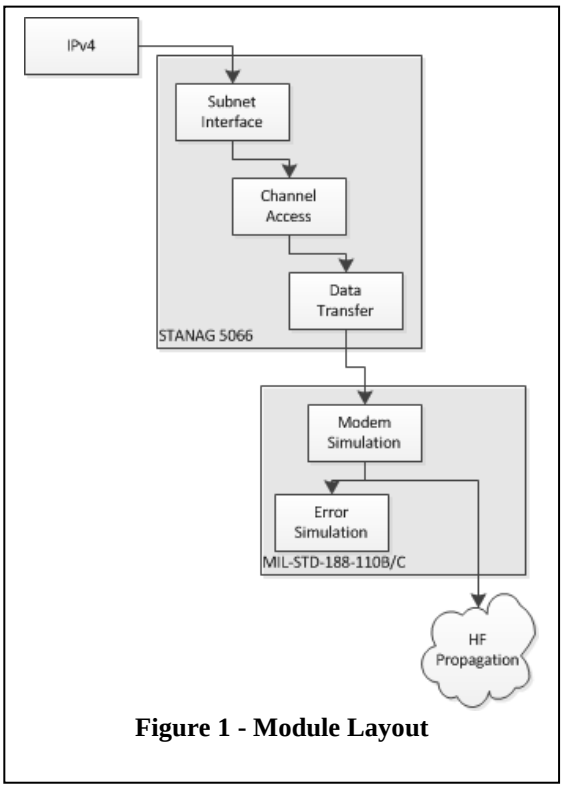


Figure 1 - Module Layout

The standard model for the HF channel has been the Watterson model with standard channel conditions defined by ITU-R Recommendation Rec. F.1487 [6]. DSP-based simulators that implement this model at the baseband level have existed for many years, and have been used to test HF modem designs. In general, these simulators use fixed values for Signal to Noise Radio (SNR), Doppler spread, and multipath delay, and simulate the signal perturbations of the HF channel quite effectively over periods of up to a few seconds [7]. However, these simulators do not model variations in signal quality that have been observed with periods of more than a few seconds to minutes.

While it would not be impossible to integrate a software implementation of this model into our simulation, the extensive signal processing computations required make them too slow for network simulations.

It is also important to note that the data rates and statistical nature of HF channel variations require much longer run times to yield valid results. A useful simulation thus must not only generate valid results, but must generate those results in much less than real time.

We have developed a software-based HF channel simulator that benefits from state-of-practice HF channel models and provides narrowband and wideband simulation capability. Modem performance statistics obtained in conjunction with this channel simulator have been used to define and calibrate the physical layer of our bearer model, as is further explained in the next section.

B. Bearer Model

The bearer part of the model consists of a modem simulation component, a data link protocol simulation component, and the internet protocol stack.

1) Modem Simulation Component

The MIL-STD-188-110B/C [3] modem simulation consists of a single simulation module which implements the functional module simulation, plus a separate C++ module which provides the propagation error simulation code (this division exists both for division of labor reasons and to allow the error simulation code to also operate as a standalone application for other uses). While it may seem more correct to place the error simulation in the HF propagation module, in fact it is more correct to say that the errors are made in the modem as it receives and decodes the incoming signal, and the pattern of errors is a function of the modem settings and the received signal plus interference.

A fundamental challenge to implementing a modem simulation is the signal processing computations required, which would adversely affect the ability to run simulations in an acceptable period of time.

The solution implemented for this simulation is to run the software-based channel and modem simulations offline to generate lengthy error traces (binary sequences containing a 1 for each simulated modem bit-error). These traces were then analyzed to determine cumulative distributions for the lengths of error-free runs and mixed runs containing errors, for each combination of channel conditions (SNR, Doppler shift, and multipath spread) and modem parameters (data rate and interleaver depth). These cumulative distributions, in table form, are then used (in the simulation framework) by a software component referred to as the 'Erracle' (Error + Oracle) to generate simulated errors having equivalent run-length distributions, so as to accurately model the 'burstiness' of the error sequences. In this manner the Erracle is capable of generating bit errors with the appropriate distribution excess of 10^7 bits per second.

The modem simulation is also responsible for determining the time required to transmit a given block of data, based on the data rate and other modem settings. Accurate transmit time information is naturally critical to ensuring that the simulation results are accurate; our modem model provides this timing information based on detailed knowledge of the modem transmission format as defined by MIL-STD-188-110B/C.

2) Data Link Protocol (STANAG 5066) Component

The STANAG 5066 [4] Simulation is broken into three submodules, corresponding to the three required functional capabilities of the protocol, SubnetInterface, ChannelAccess, and DataTransfer.

SubnetInterface is responsible for accepting IP packets from the IPv4 module at the source side of the transfer and delivering those packets at the destination side of the transfer. To simplify the lower levels of the simulation, the actual packet received is transferred via a "virtual" interface (that is, an interface that has no analog in STANAG 5066) to the destination SubnetInterface. The packet is held there until the data frame(s) that comprise the packet have been declared delivered by the lower layers of the simulation. Since the entire packet has been cached in the destination SubnetInterface, it is not necessary to actually fragment and re-assemble the packet in the lower levels, dramatically simplifying the simulation without sacrificing any simulation fidelity.

The SubnetInterface implementation is also derived into two C++ subclasses, designed to allow the exploration of different capabilities. A "Saturation" subclass ignores the network interface entirely and simply generates enough test packets to keep the interface operating at peak efficiency, to allow us to measure optimal bulk traffic throughput. This removes TCP ack and control messages, as well as making certain that sufficient traffic is always available. Theoretically, throughput measured in this mode should be a benchmark against which network throughput values can be compared, and can be used to calibrate and validate the model.

A second subclass of SubnetInterface is Acceleration, which implements the capabilities described in section III. Many parameters are exposed at this level which allows us to run simulations designed to optimize network traffic flow.

ChannelAccess is responsible for creating the HF link over which the transfer will take place. Currently, we make the assumption that the radios are already linked on a common frequency, so this module is a stub.

DataTransfer is responsible for simulating the fragmentation of packets into STANAG 5066 data frames, scheduling the transmission of those frames, verifying the proper reception of the frames, and re-assembling them into the original packets. For purposes of simulation, frames are simulated as structures that define the size and location of the data the frame represents. As noted above, the packet data is not actually fragmented or reassembled in the simulation, a substantial simplification that does not reduce the fidelity of the simulation.

3) Internet Protocol Stack

To properly simulate the network, it is necessary to simulate the internet protocol stack. In the case of the HF network, it is desirable to intercept packets at the IP level, but the behavior of the higher levels must also be taken into account.

The INET framework of OMNeT++ is ideal for our purposes, in that it includes detailed and tested simulations of TCP (in multiple variations), UDP, IP, HTTP, VOIP, and a variety of standard wireless protocols. We are indebted to the OMNeT++ community for the large and thorough base of existing work, in that it allows us to concentrate our efforts on the aspects of the simulation that are unique to HF radio.

C. Traffic Model

For our initial development, the traffic model is essentially stubbed out. We are using the TCPApp simulation component of the INET framework to generate TCP traffic of varying sizes in order to drive the simulation to produce results.

III. Performance Enhancing Proxy Overview

As expected, TCP over HF tends to work successfully only under a limited set of channel conditions. Furthermore, where it does work, it often provides limited throughput. To provide acceptable throughput, it is desirable to manage the interface a more active manner.

A TCP Performance Enhancing Proxy (PEP) [2] is used to isolate TCP from links over which it cannot be supported directly (due to error rate, latency, or other issues). The PEP intercepts IP packets as they are being routed, and masquerades as the TCP final destination to carry on the transfer with the origin TCP node. The packets are then transported through the problematic link (typically using a different protocol specifically adapted to the physical channel), where the receiving end of the PEP establishes a separate TCP connection with the real destination node, completing the transfer. TCP at both the source and destination nodes are thus isolated from the difficulties of the intermediate channel, and are provided with the illusion of a "normal" direct connection.

In the case of narrowband HF links, the performance issues to be overcome are:

1. High error rates – Under simulation, packet error rates as high as 4% have been measured on links that were able to carry TCP traffic.
2. High latency – The STANAG 5066 standard recommends transmitting for up to 120 seconds, then waiting for the acknowledgment, before transmitting again. This implies normal packet Round-Trip Times (RTT) of over 2 minutes.
3. High latency jitter – When a packet is received in error, it must be retransmitted with the next transmission. Therefore, the latency of that particular packet would typically exceed 4 minutes. In addition, as error rates rise, HF data link protocols often respond by lowering the data rate for increased robustness, further increasing the RTT.

To combat issues with the link, the PEP may locally generate TCP acknowledgments (ACKs) prior to actually delivering the data, aggregate ACKs to limit congestion, or take other actions that are required to present an interface to

the source or (less often) destination TCP implementation that conforms to expectations and allows the transfer to continue.

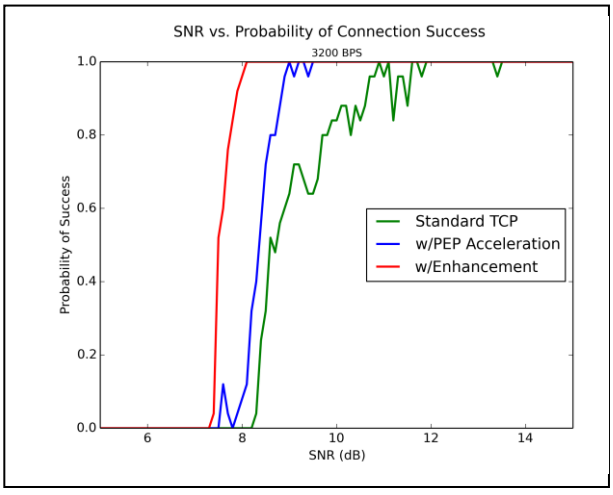
IV. Preliminary Results

The first results of this simulation were used to calibrate the model versus existing STANAG-5066 test data, to ensure that the core of the model was accurately simulating the expected behavior of the data link protocol and HF channel. Table 1 shows ideal (no-error) throughput for both the actual system and the a calculated idea throughput for various data rates, and demonstrates that the simulation does in fact accurately model STANAG-5066 at this level.

Table 1 - Model Throughput Calibration

Data Rate (bps)	Model Throughput	Calculated Throughput	Percent Difference
75	52	57	-8.8%
150	115	113	1.8%
300	228	229	-0.4%
600	465	456	2.0%
1200	890	912	-2.4%
2400	1760	1803	-2.4%
3200	2597	2543	2.1%
4800	3812	3717	2.6%
6400	4711	4536	3.9%
8000	5641	5476	3.0%
9600	6380	6202	2.9%

The next simulation series was intended to demonstrate whether and how well TCP data could be carried over HF. As previously noted, reliable and usable TCP over HF has historically been very hard to achieve with reasonable throughput. We implemented a PEP as described in section III. In addition, we have also been experimenting with various proprietary enhancements to the standard PEP strategy, which we have termed Enhanced TCP Acceleration. Figure 2 shows the probability of successful TCP transfer versus SNR for TCP, TCP with Acceleration, and TCP with Enhanced Acceleration.



The abruptness of the transitions is to be expected, both due to the fact that we are only measuring whether the transfer completed, and also due to the nature of HF modems in a static average SNR environment. More realistic long-term and medium-term variations in the Modem Model are planned. In addition, it is common to adapt the data rate of the modem to current conditions, which will likely smooth out the probability curve. This strategy will also, however, confer a throughput advantage to these enhancements.

V. Future Work

With the first version of the HF Data Network simulation working, a number of opportunities for enhancement and expansion suggest themselves;

Channel Model Improvements: It may be useful to integrate propagation predictions from a propagation model such as VOACAP directly into the simulation, simplifying the scenario configuration process.

Modem Model Improvements: It is also very likely that we will modify the channel model to take advantage of improvements in modeling such as the intermediate and long term variations in SNR described above.

Wideband STANAG 5066: We intend to modify the channel error and modem models to simulate wideband signaling. This will also require modifications to the STANAG 5066 simulation, as some of the protocol settings will not efficiently support the higher data rate transfers supported by wideband. Both the wideband signaling capability and the subsequent required modifications to STANAG 5066 are currently being considered by the NATO standardization committee, and it is anticipated that the ability to simulate these changes will provide substantial input into those discussions.

Channel Access: To realistically simulate actual HF networks, it will be necessary to simulate the process of locating an acceptable channel and creating a link with one or more other nodes on that channel. Initial efforts will simulate existing narrowband ALE protocols; later efforts will include simulation and evaluation of proposed wideband ALE technology.

Advanced Protocols: With the introduction of wideband HF, new avenues of inquiry have opened up regarding data protocols. We have also been studying so-called "Hybrid-ARQ" protocols, in which the modem and data link protocol are more closely intertwined. This allows data transfers to take advantage of diversity combining and code combining techniques to increase reliability and decrease latency. Simulation will provide us with a unique opportunity to study the performance of these techniques prior to their implementation in the target system.

Traffic Model Improvements: We anticipate that improvements to the traffic model will continue into the long term, since it is expected that each potential user environment may have its own unique traffic profiles. In the immediate future we anticipate the development of a number of representative traffic profiles to exercise the simulation in a more realistic manner.

VI. Conclusions

This simulation has been implemented in a relatively short period of time, and has already begun to produce useful and interesting results. We anticipate that the usefulness will only increase as we expand the simulation into new areas of inquiry, and presume that the output of this simulation will drive not only the development of future products, but also the development of interoperability standards worldwide.

VII. Acknowledgements

The authors gratefully acknowledge the assistance of John Nieto in the development of the HF channel model. The authors would also like to thank the OMNeT++ community for the development of this toolset and for the documentation and on-line assistance that is available, all of which were of great assistance in the development of this model.

References

[1] E. Johnson, E. Koski, W. Furman, M. Jorgenson, and J. Nieto, Third-Generation and Wideband HF Radio Communications. Boston, Artech House, 2012. ISBN-13: 978-1608075034.

[2] Perfomance Enhancing Proxies Intended to Mitigate Linik-Related Degredations, IETF RFC 5135. Available: http://tools.ietf.org/html/rfc3135

[3] MIL-STD-188-110C, "Interoperability and Performance Standards for Data Modems", US Department of Defense, 23 September 2011. Available: http://hflink.com/standards/MIL_STD_188-141C.pdf.

[4] STANAG 5066, "Profile for High Frequency (HF) Radio Data Communications," Edition 3 (Ratification Request), North Atlantic Treaty Organization, 2010.

[5] E. Koski, S. Chen, S. Pudlewski, and T. Melodia, "Network simulation for advanced HF communications engineering," Ionospheric Radio Systems and Techniques (IRST 2012), 12th IET International Conference, 15-17 May 2012. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6251384&isnumber=6250647

[6] ITU-R Rec. F.1487. "Testing of HF Modems with Bandwidths of up to About 12 kHz Using Ionospheric Channel Simulators." International Telecommunication Union, Radiocommunication Sector. Geneva, 2000.

[7] W.N. Furman and E. Koski, "Standardization of an intermediate duration HF channel variation model," Ionospheric radio Systems and Techniques, 2009. (IRST 2009). The Institution of Engineering and Technology 11th International Conference on, pp.1-5, 28-30 April 2009.

Skip This Paper

RINASim: Your Recursive InterNetwork Architecture Simulator

Vladimír Veselý, Marcel Marek, Tomáš Hykel, Ondřej Ryšavý
Department of Information Systems, Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
{ivesely, imarek, rysavy}@fit.vutbr.cz; xhykel01@stud.fit.vutbr.cz

***Abstract*—Recursive InterNetwork Architecture is a clean-slate approach to how to deal with the current issues of the Internet based on the traditional TCP/IP networking stack. Instead of using a fixed number of layers with dedicated functionality, RINA proposes a single generic layer with programmable functionality that may be recursively stacked. We introduce a brand new framework for modeling and simulation of RINA that is intended for OMNeT++.**

Keywords—Recursive InterNetwork Architecture; RINA; OMNeT++; Delta-t;

I. Introduction

Thank you for not skipping this paper and now straight to the point. RFC 4984 [1] and a follow-up RFC 6227 [2] mention some of the current issues of the Internet. Cumbersome mobility and multihoming (mostly due to the fact of incomplete naming scheme where IP address identifies an interface, not the device), missing native Quality of Service (QoS) support, prefix deaggregation and overall unscalable growth of the default-free zone (backbone of the Internet) are among these problems. Since 2006, many alternatives have been extensively discussed and investigated (e.g., Locator/ID Split Protocol, Identifier-Locator Network Protocol, Multipath TCP, etc.). However, none of them were able to come up with a unifying solution to all problems.

Recursive InterNetwork Architecture (RINA) is a complete replacement of the traditional TCP/IP (or ISO/OSI) layered network model based on a rigid differentiation between functionalities of each layer. RINA (initially proposed by John Day's book [3]) is a subject of ongoing research interest and initiatives during last few years. RINA's core principle says that *networking* is nothing more than only inter-process communication (IPC). The IPC happens over different scopes, where the layer limits a scope. Layers may be recursively stacked to provide IPC services to applications and/or layers above. Layer in RINA is called **Distributed IPC Facility (DIF),** and it is formed by a set of cooperating IPC processes. Each DIF is completely independent, offering wide gamut of mechanisms in order to achieve: (un)reliable data transfer; flow and congestion control; relaying and multiplexing of traffic; routing decisions or other DIF management tasks; etc. RINA clearly distinguishes between mechanisms and policies, which means that all DIFs offer the same set of mechanisms but operate under potentially different configurations (policies).

The purpose of this paper is to present the community with RINASim – the RINA-capable framework for the OMNeT++ simulator – and describe its components and basic order of operations. This paper has the following structure. The next section covers a brief introduction to some of RINA concepts. Section III reveals components of our framework together with their description. Section IV describes flow lifecycle in two simple scenarios where a pair of hosts is exchanging data. The paper is summarized in Section V together with unveiling of our plans.

II. State of the Art

This section introduces basic RINA principles and terminology. However, astute reader is advised to consult additional information sources (see [4], [5] or [6]) because not everything can be explained due to the limited space of this paper.

Nature of *applications* in RINA is as follows: **Application Process (AP)** is program instantiation to accomplish some purpose; **Application Entity (AE)** is the part of an AP, which represents the application protocol and application aspects concerned with the IPC. There may be multiple instances of the Application Process in the same system. AP may have multiple AEs, each one may process a different application protocol. There also may be more than one instance of each AE type within a single AP. All application protocols are stateless; the state is and should be maintained in the application. Thus, all **application protocols** modify shared state external to the protocol itself on various objects (e.g., data, files, HW peripherals). Because of that, there is only one application protocol that contains trivial operations (e.g., read/write, start/stop, and create/delete).

As it was mentioned before, RINA separates mechanisms (fixed parts of a system) from policies (variable parts of a system) of any IPC. There are two types of mechanisms: a) **tightly-bound** that must be associated with every packet, which handle fundamental aspects of data transfers; and b) **loosely-bound** that may be associated with packet and which provide additional features (namely reliability and flow control). Both types are coupled through a state vector maintaining state information. Previous implies the existence of a single soft state **data transfer protocol**. This protocol controls data transfer with the help of different policies. Initial synchronization of communicating parties is done with the help of **Delta-t** protocol

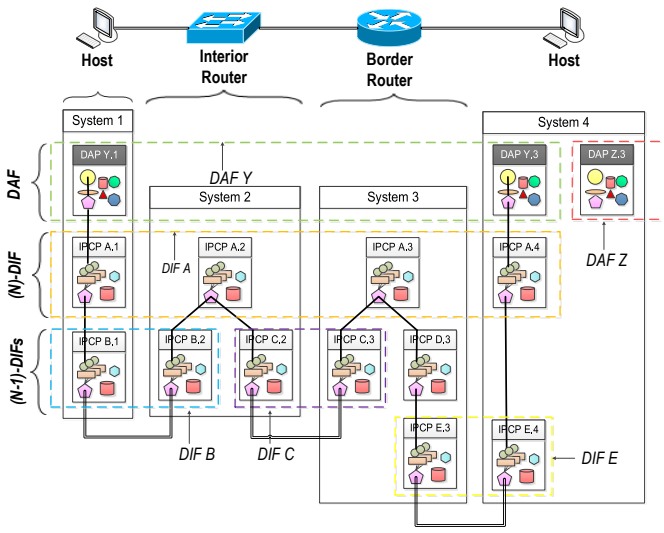


Fig. 1. DIF, DAF, DAP and IPCP illustration

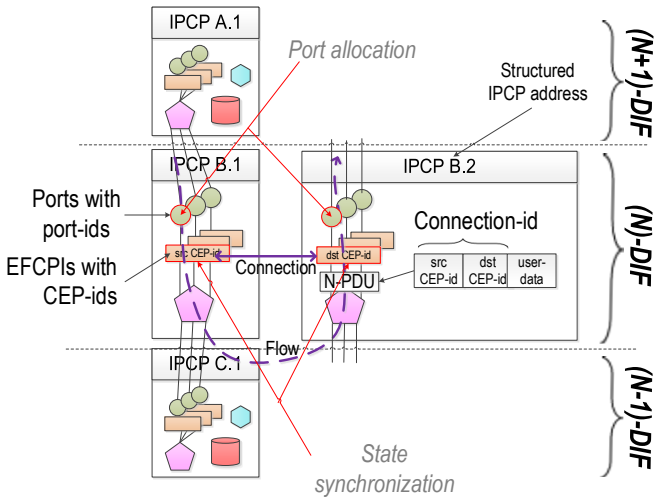


Fig. 2. IPCP's local identifiers overview

see ([7], and[8]). Delta-t was developed by Richard Watson, who proposed a timer-based synchronization technique. He proved that conditions for distributed synchronization are met if following three timers are bound: a) Maximum Packet Lifetime (MPL); b) Maximum time to attempt retransmission a.k.a. maximum period during which a sender is holding a packet for retransmission when waiting for a positive acknowledgement (R_{timer}); c) Maximum time before Acknowledgement (A_{timer}). Delta-t assumes that all connections exist all the time. The synchronization state is maintained only during the data transfer activity, but after $2\Delta t$ or $3\Delta t$ periods without any traffic the state may be discarded which effectively resets the connection, where $\Delta t = MPL + A_{timer} + R_{timer}$. Because of that, there is no need for hard state (with explicit synchronization using SYNs and FINs). Delta-t postulates that port allocation and synchronization are distinct.

The concept of RINA layer a.k.a. DIF could be further generalized to **Distributed Application Facility (DAF)** – a set of cooperating APs in one or more computing systems, which exchange information using IPC and maintain shared state. A DIF is a DAF that does only IPC. **Distributed Application Process (DAP)** is a member of a DAF. **IPC Process (IPCP)** is a special AP within a DIF delivering inter-process communication. IPCP is an instantiation of a DIF membership; computing system can perform IPC with the other DIF members via its IPC process within this DIF. An IPCP is specialized DAP. The relationship between all newly defined terms is depicted in Fig. 1.

We need to differentiate between different APs and also different AEs within the same AP. Thus, RINA uses **Application Process Name (APN)** as a globally unambiguous, location-independent, system-dependent name. **Distributed Application Name (DAN)** is a globally unambiguous name for a set of system-independent APs. An IPC Process has an APN so it can be identified among other DIF members. An **address** is a synonym for the IPCP's APN with a scope limited to the layer and structured to facilitate forwarding. An APN is useful for management purposes but not for forwarding. The address structure may be topology-dependent (indicating the nearness of IPCPs). An APN and an address are simply two different means to locate an object in different context. There are two local identifiers important for the IPCP functionality – a port-id and a connection-endpoint-id. **Port-id** binds this (N)-IPCP and (N+1)-IPCP/AP; both of them uses the same port-id when passing messages. Port-id is returned as a handle to the allocator and is unambiguous within a computing system. **Connection-endpoint-id (CEP-id)** identifies a shared state of one communication endpoint. Since there may be more than one flow between the same IPCP pair, it is necessary to distinguish them. For this purpose, **Connection-id** is formed by combining source and destination CEP-ids with QoS requirements descriptor. CEP-id is unambiguous within an IPCP and the Connection-id is unambiguous between a given pair of IPCPs. Fig. 2 depicts all relevant identifiers between two IPCPs. Watson's Delta-t implies Port-id and CEP-id to help separate port allocation and synchronization. The RINA **connection** is a shared state between protocol machines – ends identified by CEP-ids. The RINA **flow** is bound to ports identified by port-ids. The lifetimes of a flow and its connection(s) are independent of each other.

III. Contribution

We are developing **RINASim** in the frame of FP7 project PRISTINE [9]. The goal is to provide the public community with a full-fledged RINA simulator to support ongoing research and academic activities. To understand RINA architecture means to understand each of its elements. This subsection starts with a description of high-level RINA network nodes and then goes deeper and outlines various implemented components.

There are only three basic kinds of nodes in RINA network. Each kind represents computing system running RINA: a) **hosts**, IPC end-devices containing AEs, they employ two or more DIF levels; b) **interior routers**, interim devices interconnecting (N)-DIF neighbors via multiple (N-1)-DIFs, they employ two or more DIF levels; c) **border routers**, interim devices interconnecting (N)-DIF neighbors via (N-1)-DIFs, where some of (N-1)-DIFs are reachable only through (N-2)-DIFs, they employ three or more DIF levels. Fig. 1 depicts simple RINA

network containing all kinds of nodes and their basic internal structure.

The internal structure of any RINA node could be divided into two parts – the one responsible for DAF operation and another one for DIF operation. DAF part contains one or more APs and a couple of management components, which are described in Table I. DIF part contains one or more IPCPs of different ranks interconnected to create a recursive stack. All IPCPs consist of a same set of subcomponents, which are depicted in Fig. 3 and summarized in Table II.

TABLE I. DAF COMPONENTS OVERVIEW

Name	Description
`application Process`	AP module contains one or more AE instances. AE processes application protocol and uses IPCP to communicate with destination AE(s).
`ipcResource Manager`	IPC Resource Manager (IRM) manages DAF resources, which involve delegation of flow (de)allocation calls or management of a new DAF/DIF join or creation. IRM maintains information about all connections used by AE(s).
`dif Allocator`	DIF Allocator (DA) primary task is to return a list of DIFs where destination application may be found given APN and access control information. DA contains and works with multiple mapping tables to provide its services.

RINA's generic approach to an IPCP's operation is met via NED interfaces. Each policy is represented as a simple module implementing specific interface. NED interfaces enabled us to use different implementations for a specific policy, which is not possible to do directly in C++. Policies related to static modules (RMT, FA) are specified via variables in a simulation configuration file (.ini) and policies related to dynamic modules (EFCP instances) are configured by an XML configuration file. Dynamic modules cannot be addressed in the .ini file before the start of the simulation because their names are not yet known due to randomness in generated names.

All inner IPCP interconnections are modeled with zero-time delay since processing messages between modules is a matter of a queue scheduling algorithm and not a network architecture. The bottommost interconnection between IPCPs represents the physical medium offering to set various properties (rate, delay, bit error rate). The recursiveness enables us to set the properties on all interconnections between IPCPs.

TABLE II. IPCP COMPONENTS OVERVIEW

Name	Description
`flow Allocator`	Flow Allocator (FA) processes (de)allocate calls. FA creates a Flow Allocator Instance (FAI), which manages each flow independently. FAI spawns separate EFCP instance to handle data and interconnects all involved components with bindings to create data-path. FA translates application QoS requirements onto available RA's QoS profiles.
`efcp`	Error and Flow Control Protocol (EFCP) provides data transfer services, and it is split into two independent parts. Data Transfer Protocol (DTP) implements mechanisms tightly coupled with transported SDUs, e.g., fragmentation, reassembly, and sequencing. DTP works with packet header fields like source/destination addresses, QoS requirements, Connection-id,
	optionally sequence number or checksum. Data Transfer Control Protocol (DTCP) implements mechanisms that are loosely coupled with transported SDUs, e.g., (re)transmission control using various acknowledgment schemes and flow control with data-rate limiting. DTCP functionality is based on Delta-t.
`relay And Mux`	Relay and Multiplexing (RMT) instances in hosts and bottom layers of routers usually perform just the multiplexing task, while RMTs in top layers of interior/border routers do both multiplexing and relaying. RMTs in top layers of border routers perform aggregation of traffic from multiple (N)-flows onto single (N-1)-flow. Each (N-1)-port handled by RMT has own set of input and output buffers. RMT performs routing decision putting PDUs into proper queues of (N-1)-flows leading to various destinations.
`resource Allocator`	If a DIF has to support diverse QoS, then different flows will have to be allocated to different policies and traffic for them treated differently. Resource Allocator (RA) is component accomplishing this goal by monitoring and managing RMT queues, EFCP policies, available QoS profiles, etc.
`ribDaemon`	All management information maintained by IPC components such as FA, RA, etc. is available and updated through RIB Daemon (RIBd). RIBd is sending and receiving management message on behalf of other components.
`routing Policy`	This module is pure policy computing an optimal path to other destinations by given metrics. Usually some routing algorithm exchanges information with other IPCP members of a DIF.
`enroll ment Module`	Enrollment module is involved when IPCP is joining a DIF. At this time, Enrollment module finite-state machine governs the exchange of authentication information during the connection establishment phase.

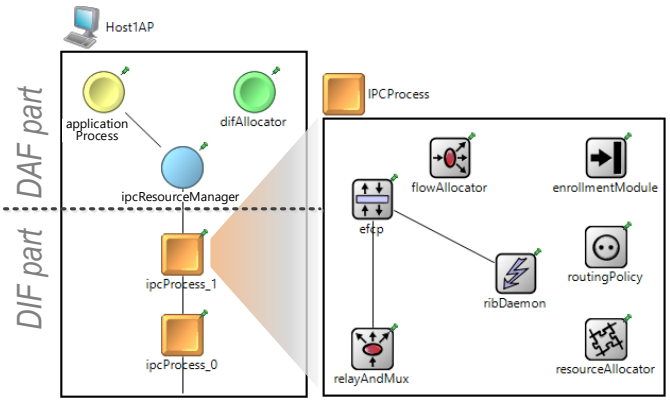


Fig. 3. Host node and IPC Process structure

IV. TESTING

In this section, we present example scenario to illustrate the flow lifecycle. Simulation topology (shown in Fig. 4) consists of two hosts (`Host1` and `Host2`) and one interior router (`SW`) performing relaying between the two hosts. Hosts APs `AP_A` and `AP_B` employ a ping-like application protocol exchanging request-response messages and measure one-way/round-trip time latencies.

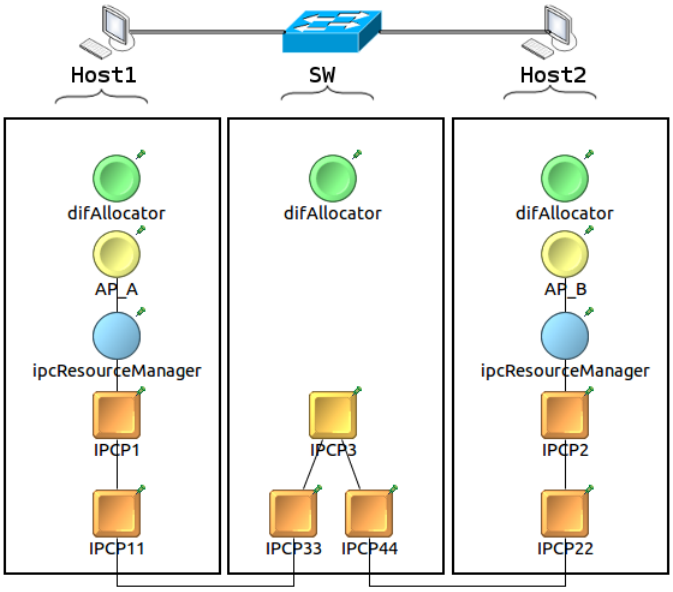


Fig. 4. Testing scenario

The flow initialization can be separated into five phases. The odd phases belong to `Host1`, and even phases belong to `Host2`. The first phase describes beginning of the allocation.

1) `AP_A` on `Host1` notifies IRM with its request;
2) IRM notifies `IPCP1`'s FA to check whether there is any flow available between applications `A` and `B`. Since there is neither data nor management (N)-flow, `IPCP1`'s FA invokes management flow allocation in order to create IPC channel for exchange of management messages;
3) `IPCP1`'s FA asks `IPCP1`'s RA to prepare (N-1)-flow for management. Flow allocation process is recursively repeated from the beginning whenever there is no available flow in the underlying DIF until it reaches medium, where flow allocation is considered inherent.

During the second phase, `IPCP2`'s FA of `Host2` receives allocation request, which is accepted. Subsequently, bindings between bottom `IPCP22` and `IPCP2`'s RIBd are formed.

During the third phase, `Host1` `IPCP1`'s FA receives a positive allocation response for management flow and starts enrollment procedure. During enrollment `Host1` and `Host2` exchanges authentication data. Upon successful enrollment, `Host1` continues initial data flow allocation.

1) `Host1`'s FA creates FAI for data flow. FAI spawns EFCP instance. Following next, FAI prepares bindings and lets RIBd send *CreateFlowRequest*;
2) *CreateFlowRequest* is processed by `IPCP3`'s FA where it needs to be forwarded to `IPCP44`. But before that underlying (N-1)-flow are recursively allocated, which is secured by `IPCP3`'s RA.

During the fourth phase, `Host2` handles *CreateFlowRequest* for data flow.

1) `IPCP2`'s RIBd notifies `Host2`'s AP about flow allocation and connection attempt. `AP_B` accepts and governs IRM to prepare bindings and delegate flow allocation request;
2) IRM invokes `IPCP2`'s FA, which instantiates FAI. FAI spawns EFCP instance to provide data transfer service to the data flow. FAI prepares bindings and lets RIBd send positive *CreateFlowResponse*.

During the fifth phase, `IPCP1`'s RIBd receives a positive *CreateFlowResponse* and notifies about it FAI. Then, FAI notifies the application about the successful finalization of the flow allocation. Subsequently, the actual ping-like data transfer occurs between and using allocated data-paths.

When APs finish data exchange, the flow deallocation process is initiated. Deallocation is separated into three phases.

During the first phase, `AP_A` initiates data flow deallocation.

1) Before actual deallocation, AP `A` releases connection notifying `AP_B` about this fact;
2) Following next, `AP_A` tells IRM about deallocation request, which delegates it to appropriate `IPCP1`'s FAI;
3) `IPCP1`'s FAI asks RIBd to deliver *DeleteFlowRequest*.

During the second phase, `AP_B` receives dellocation request.

1) `IPCP2`'s FAI on `Host2` receives *DeleteFlowRequest*. It notifies `AP_B` about it, and initiates data flow deallocation by removing appropriate EFCP instance and disconnecting bindings;
2) RIBd of `IPCP2` acknowledges deallocation by sending to `IPCP1` *DeleteFlowResponse* on behalf of FAI. At this point, data flow and data-path from `AP_B` side on `Host2` are deallocated.

During the third phase, deallocation is finalized on `AP_A` side. `IPCP1`'s FAI receives a *DeleteFlowResponse* and finishes data flow deallocation by removing EFCP and relevant bindings. Deallocation is complete, and port-IDs associated with `AP_A` and `AP_B` may be reused by other APs.

Simulation results and message confluence have been successfully verified against proposed behavior in RINA specification.

V. Conclusion

We have outlined the basic principles behind RINA, a clean-slate replacement of the traditional TCP/IP stack. We have introduced RINASim as an independent OMNeT++ framework providing simulation environment for educational and research purposes with this fresh architecture. We plan to carry on work and further refine our framework based on new knowledge and up-to-date specifications. An additional goal is to conduct a comparative evaluation of our simulation models with RINA implementation for Linux environment called IRATI [10]. All source codes are publicly available on GitHub repository [11] under the MIT license. We encourage the reader to read accompanied documentation [12] or generate Doxygen documentation to get more insight.

ACKNOWLEDGMENT

This work was supported by the Brno University of Technology organization and by the research grants: FP7-PRISTINE supported by European Union; FIT-S-14-2299 supported by Brno University of Technology; IT4Innovation ED1.1.00/02.0070 supported by Czech Ministry of Education Youth and Sports.

REFERENCES

[1] D. Meyer, L. Zhang, and K. Fall. (2007, September) RFC 4984:Report from the IAB Workshop on Routing and Addressing. [Online]. http://tools.ietf.org/html/rfc4984

[2] T. Li. (2011, May) RFC 6227: Design Goals for Scalable Internet Routing. [Online]. http://tools.ietf.org/html/rfc6227

[3] J. Day, *Patterns in Network Architecture: A Return to Fundamentals*, 1st ed. ISBN-13: 978-0137063383: Prentice Hall, 2008.

[4] J. Day, I. Matta, and K. Mattar, "Networking is IPC: a guiding principle to a better internet," in *Proceedings of the 2008 ACM CoNEXT Conference*, NY, USA, 2008, p. 6.

[5] E. Trouva, E. Grasa, J. Day, and S. Bunch, "Layer discovery in RINA networks," in *IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks(CAMAD)*, Barcelona, Spain, 2012, pp. 368 - 372.

[6] E. Trouva et al., "Is the Internet an unfinished demo? Meet RINA!," in *TERENA Networking Conference*, Prague, Czech Republic, 2011, p. 12.

[7] R. Watson, "Delta-t protocol specification," Lawrence Livermore National Lab., California, USA, UCID-19293, 1981.

[8] R. Watson, "The Delta-t transport protocol: features and experience," in *Proceedings 14th Conference on Local Computer Networks*, Mineapolis, MN, USA, 1989, pp. 399-407.

[9] PRISTINE consortium. (2015) PRISTINE | PRISTINE will take a major step forward in the integration of networking and distributed computing. [Online]. http://ict-pristine.eu/

[10] F. Salvestrini. (2015) IRATI · GitHub. [Online]. https://github.com/IRATI

[11] Brno University of Technology. (2015) kvetak/RINA · GitHub. [Online]. https://github.com/kvetak/RINA

[12] V. Veselý. (2015) RINA Simulator: basic functionality. [Online]. http://ict-pristine.eu/wp-content/uploads/2013/12/PRISTINE-D24-RINASim-draft.pdf

Implementation of a Wake-up Radio Cross-Layer Protocol in OMNeT++ / MiXiM

Jean Lebreton and Nour Murad
University of La Reunion, LE2P
40 Avenue de Soweto, 97410 Saint-Pierre
Email: jean.lebreton@univ-reunion.fr

***Abstract*—This paper presents the DoRa protocol, which is a new cross-layer protocol for handling the double radio of nodes in wake-up radio scenario. The implementation details in OMNET++/MiXiM are also given, with a focus on the implemented MAC layers. The main goal of the DoRa protocol is to reduce energy consumption in wireless sensor network, by taking full advantage of the passive wake-up scheme. The performance of the DoRa protocol is then evaluated and results are compared with B-MAC and IEEE 802.15.4 protocols.**

I. Introduction

Energy conservation in Wireless Sensor Networks (WSN) is essential since nodes are usually powered by limited batteries. Several approaches have been considered in the literature to reduce the energy consumption of the network [1]. Duty cycling mechanisms are part of the proposed approaches and they can prolong the nodes lifetime consequently. In duty cycling mechanisms, nodes alternate between different states (typically awake and sleep) in order to avoid idle listening, which is the major source of energy depletion in WSN. We propose to reduce even more the duty cycle with a new wake-up radio cross-layer protocol, which means the nodes would wake up on demand by a particular radio message.

A second radio module is plugged into the nodes for the wake-up communications and the main radio module is used for data transmission. The second radio, also known as wake-up radio, must be a low-power device or completely passive. Few MAC protocols were proposed during the last years to incorporate the wake-up radio features, such as RTM [2], On Demand MAC [3] and SCM-MAC [4]. These protocols reduce significantly the energy consumption of the nodes but they do not take full advantage of the wake-up scheme. Our MAC protocol takes into account the fact that communication is done while other nodes are sleeping, enabling the non-use of RTS/CTS mechanisms or acknowledgements.

This paper presents the implementation of a new MAC protocol in OMNeT++ 4.6 with the MiXiM 2.3 framework [5],[6]. The node architecture has been modified to enable the communication with two radio modules and cross-layer information exchanges, as the common node model is not appropriate for our proposed MAC protocol. Besides, the B-MAC [7] and IEEE 802.15.4 MAC protocol [8] are already implemented in MiXiM, which enables the performances comparison with existing protocols.

Fig. 1. Frame structure of exchanged messages between the BS and nodes.

II. DoRa protocol design

The DoRa protocol is designed for hierarchical networks composed of a Base Station (BS) and N nodes, in which a one-hop communication is used between the BS and nodes. The BS is in charge of every wake-up procedures and it triggers the nodes wake-up one at a time, which prevents potential communication conflicts if they would wake themselves up. This mechanism is more energy-efficient from nodes perspective since they can directly communicate after waking up. Some hardware modifications are required on the nodes to be compliant with the DoRa protocol. A second low-power micro-controller is used with a passive radio receiver, which can harvest energy from received signals. These two main modules defines our Wake-up Radio (WuR).

All nodes are initially in sleeping mode with their main radio card switched off. The BS sends a wake-up call periodically to each node by including its MAC address in the wake-up call message. The awake node answers by sending the data message directly to the BS. Both frame structures are depicted in figure 1. Before sending a new wake-up call, the BS waits during a predefined timeout window T_{out} or until the data reception from the concerned node. Every node is requested with this communication scheme and no internal interference is possible since nodes stay in sleep mode while they do not receive a wake-up call addressed to them. The communication principle is illustrated in figure 2.

This protocol implies that the BS alternates between transmission and reception mode. The reception state is activated once a wake-up call is sent and the transmission states is activated when the wake-up call must be sent. Concerning the node, its main radio has to switch between sleeping mode and transmitting mode, the latter being activated after the wake-up radio has confirmed a wake-up call. Two main criteria must be fulfil for confirming a wake-up call in our scheme. First, the received power of a signal should be strong enough all along the wake-up preamble duration, corresponding to a continuous

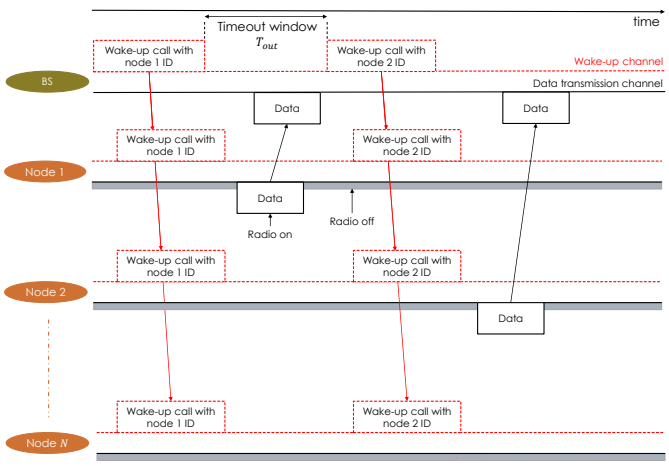


Fig. 2. Communication principle with the DoRa protocol.

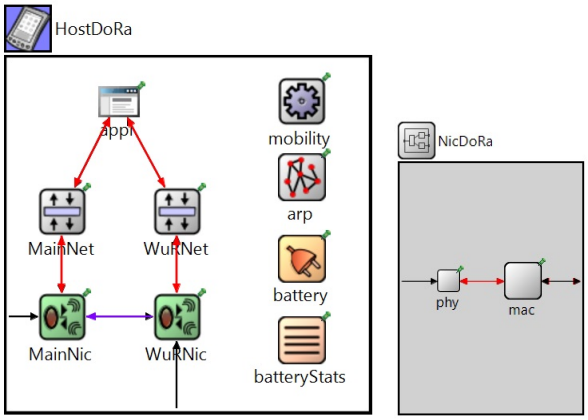


Fig. 3. Structure of *HostDoRa*.

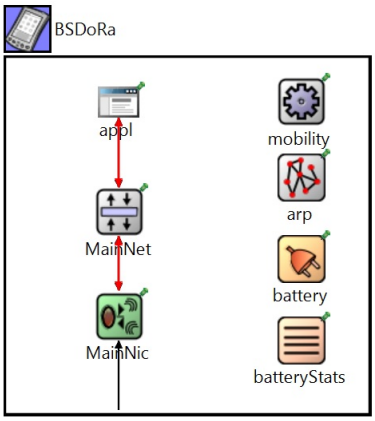


Fig. 5. Structure of *BSDoRa*.

voltage delivered by a passive wake-up circuit. The second condition is the validation of the MAC address included in the wake-up call message.

III. Implementation

This section provides the implementation details of the DoRa protocol in OMNeT++/MiXiM. The node and BS modules are different in our simulation since they have distinct architectures and features. The main contribution lies in the MAC layers used by each module. After describing the main implementation, the finite state machine of each MAC layer is presented.

A. Node module

The node module is called *HostDoRa* and its overall structure is shown in figure 3. The major implementation is done in both MAC layers, namely *Mac_Main* for handling the communication with the main radio and *DoRaMacLayer* for handling the communication associated with the wake-up process.

- The *DoRaMacLayer* handles wake-up calls sent by the BS. The state machine of the micro-controller, which is in charge of the wake-up procedure, is implemented in this layer. After processing and validating the destination address in a wake-up call, *DoRaMacLayer* sends a control message to both the other MAC layer (*Mac_Main*) and application layer for sending data to the BS.

- The *Mac_Main* handles the state machine of the main radio module. After receiving a control message from *DoRaMacLayer*, it goes to *idle* state and waits for the message from the application layer. The radio is then switched to *Tx* state when the data is received. After transmitting the data through the medium, a control message is sent to *DoRaMacLayer* and the node goes back to *sleep* state.

- A new gate is defined between the MAC layers, enabling a cross-layer communication for control messages. This gate is defined in a new basic module called *BaseLayerCross*. This module is an extension of the *BaseLayer* module, which is the basic layer for every module. There are currently 2 lower gates and 2 upper gates, for data or control messages with the lower or upper layers. We added a side gate for sending control message to an other layer, with the associated function for sending message through the gate. The control messages could be passed through the network and application layers, but a direct communication between the two MAC layers is preferred for saving time in the wake-up procedures. Taking less time for waking up the nodes or going to sleep mode also reduces the energy consumption, which is the main motivation of this work.

- Some modifications were done in the physical layer of MiXiM, especially in the decider. We derived the *Decider802154Narrow* for creating the *DeciderWakeUpRadio* in which a signal is processed if the signal duration is long enough, and with a sufficient energy level all along the signal duration. This type of signal corresponds in fact to our wake-up preamble, which is a distinctive feature from data messages.

B. Base Station module

The BS module is called *BSDoRa* and its structure is given in figure 5. Only one radio module radio is used for both the

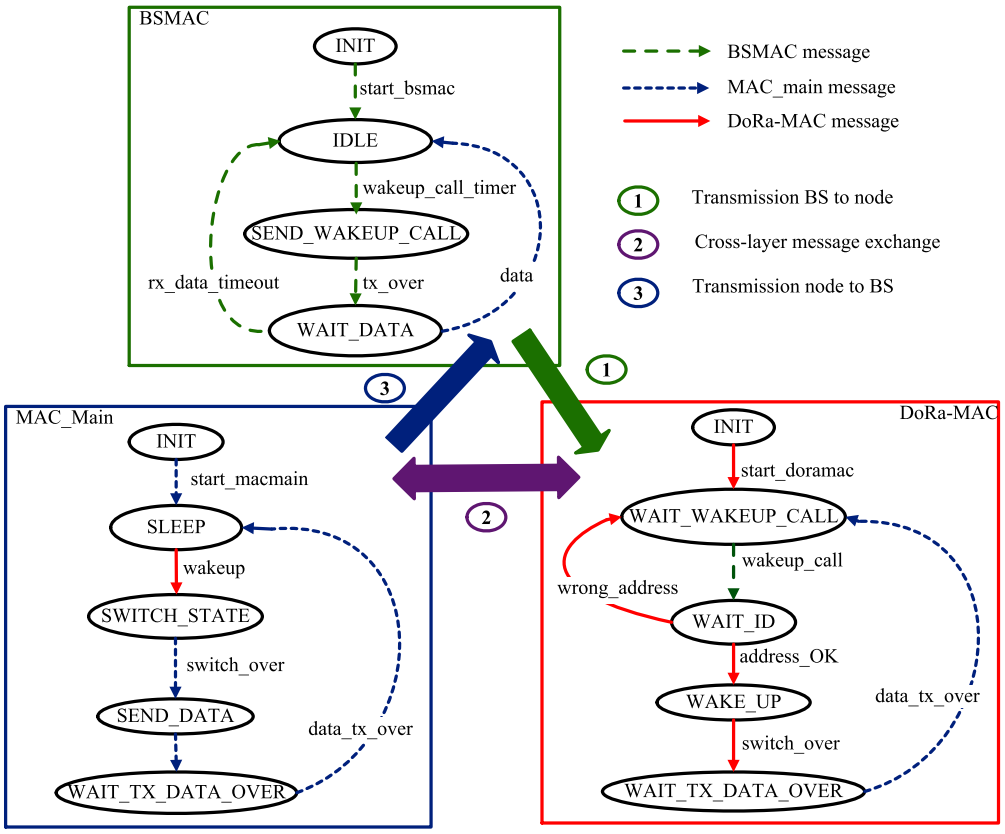


Fig. 4. Finite state machine of each MAC layer of DoRa protocol.

wake-up call and the data reception, since they are delivered in the same channel. A MAC layer called *BSMAC* have been implemented for handling the switching between wake-up call transmission and data reception, according to the DoRa protocol. The *BSMAC* is also in charge of incrementing the destination address included in the wake-up call message in order to request data from every nodes.

C. Finite state machines

The finite state machine of each MAC layer is given in figure 4 and shows the required interactions. Within the node, the interactions between the two MAC layers is the reason of our cross-layer approach. All MAC modules are initialized at the same time and they switch to their first state, waiting for an event to continue. Upon the end of the wake-up call timer, *BSMAC* sends a wake-up call message in the related state. After transmission, the module *BSMAC* waits for data coming from the *MAC_main* module.

The *DoRa-MAC* module switches its state for decoding the MAC address when it receives the wake-up call message. If the destination address included in the message corresponds to its own MAC address, then *DoRa-MAC* switches to *WAKE_UP* state where it sends the control message called *wakeup* to the *MAC_main* module. The last state consists of waiting for the completion of data transmission from the *MAC_main* module.

The *MAC_main* module leaves the *SLEEP* state when it receives the *wakeup* control message, for switching its radio to transmitting mode. After sending data through the channel, the module goes back to *SLEEP* state. At this stage, all modules are back to their first states, since a complete cycle of data request and data transmission is done.

IV. Performance evaluation

The DoRa protocol is compared with the B-MAC and IEEE 802.15.4 protocols through the same simulation scenario. Even if B-MAC and IEEE 802.15.4 are chosen due to their existing implementation in MiXiM, it gives an interesting comparison of our on-demand wake-up protocol with a duty-cycle approach (B-MAC) and an always on radio approach (IEEE 802.15.4).

A network comprised of a BS and 5 nodes is defined and every node sends data periodically to the BS, with a direct transmission. The inter packet arrival time is the time between two consecutive packets reception and it characterizes the traffic rate. The variation of the inter packet arrival time enables the performance evaluation of the network for each protocol. The other simulations parameters are set with a constant value and the main parameters are given in table I.

A number of 8 simulations are executed by incrementing the inter packet arrival time with each protocol. Even if the performance can be evaluated through several statistics, we

TABLE I
SIMULATION PARAMETERS

Protocol	Parameter	Value
Common	Surface area	20 m × 20 m
	Number of nodes	5
	Packet payload	100 bytes
	Queue buffer length	10 packets
	Supply voltage	3.3 V
	Battery capacity	2900 mAh
	Bit rate	250 kbps
	Sleep current	900 nA
	Rx current	16.6 mA
	Tx current	21.2 mA
IEEE 802.15.4	Backoff number	3-5
	CCA detection time	128 μs
B-MAC	Slot duration	1 s
	Check interval	10 ms
DoRa-MAC	Sleep current (WuR)	200 nA
	Rx current (WuR)	1.1 μA
	Active current (WuR)	9 mA
	Bit rate (WuR)	32 kbps
	Timeout window	10 ms

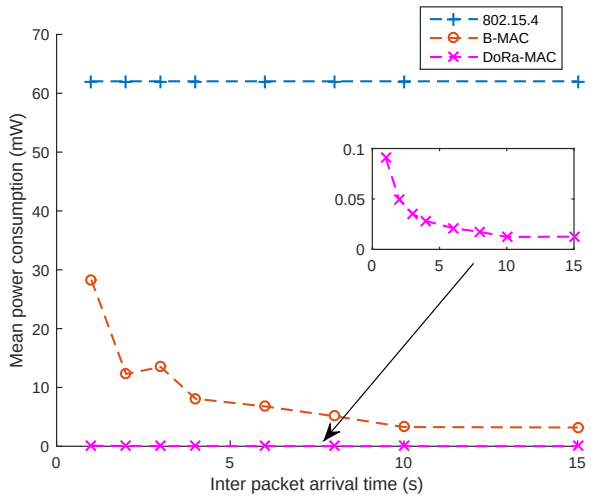


Fig. 6. Mean power consumption versus the inter packet arrival time for each protocol.

focus our study in the mean power consumption of the nodes and packet delivery ratio at the BS.

The mean power consumption is defined as the ratio between the total amount of consumed energy and the total operation time. This mean average power consumption is averaged for every node and its evolution is depicted in figure 6. As no duty-cycle is defined in the IEEE 802.15.4 protocol, the mean power consumption remains constant while varying the inter packet arrival time. The power consumption is relatively high because of continuous idle listening. B-MAC give better results since it takes advantage of duty-cycle for saving noticeable energy compared to IEEE 802.15.4. As depicted in the figure 6, the DoRa protocol outperform the others since the mean power consumption is reduced significantly. This graph also shows that the DoRa is even more suitable for application with low traffic rate, even if it still performs well with higher traffic rate.

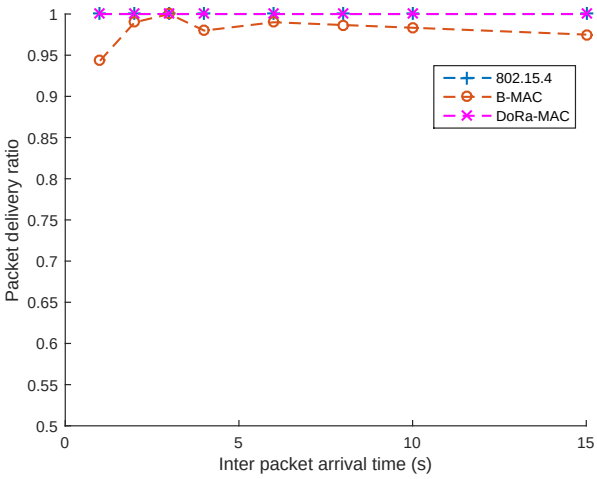


Fig. 7. Packet delivery ratio versus the inter packet arrival time for each protocol.

The results presented in figure 6 can also be used to give the amount of energy saving for B-MAC and DoRa-MAC compared to IEEE 802.15.4. Considering an inter packet arrival time of 15 seconds, B-MAC saves up 95% of energy while DoRa-MAC saves up 99.98% of energy, both compared to 802.15.4 MAC protocol.

The figure 7 gives the packet delivery ratio at the BS, defined as the number of packet received at the BS divided by the number of packets generated by the nodes. The results are consistent for the IEEE 802.15.4 and DoRa protocol since the packet delivery ratio is always equal to 1, while it is comprised between 0.94 and 1 for the B-MAC protocol. This graph shows the reliability of our DoRa-protocol since all the packets are received.

V. CONCLUSION

The main motivation of the DoRa protocol is to reduce energy consumption in WSN, by providing a cross-layer protocol for passive wake-up radio architecture. The DoRa protocol is presented and its implementation is described in this paper. Simulations were carried out in order to evaluate the performance of DoRa protocol in comparison with B-MAC and IEEE 802.15.4 protocols. The results clearly show the benefit of using DoRa compared to both B-MAC and IEEE 802.15.4 since it reduces significantly the average energy consumption. The results also show the huge benefits of using a passive wake-up radio architecture compared to an always on radio or duty-cycle approach.

REFERENCES

[1] T. Rault, A. Bouabdallah, and Y. Challal, "Energy efficiency in wireless sensor networks: A top-down survey," *Computer Networks*, vol. 67, pp. 104–122, July 2014.

[2] P. Sthapit and J.-Y. Pyun, "Effects of Radio Triggered Sensor MAC Protocol over Wireless Sensor Network," in *11th IEEE International Conference on Computer and Information Technology*, pp. 546–551, IEEE, 2011.

[3] M. A. Ameen, U. Niamat, C. M. Sanaullah, and K. Kyungsup, “A MAC Protocol for Body Area Networks using Out-of-Band Radio,” in *Wireless Conference 2011 - Sustainable Wireless Technologies (European Wireless), 11th European*, (Berlin; [Piscataway, N.J.]), VDE Verlag ; IEEE, 2011.

[4] J. Oller, I. Demirkol, J. Casademont, J. Paradells, G. U. Gamm, and L. Reindl, “Wake-up radio as an energy-efficient alternative to conventional wireless sensor networks MAC protocols,” in *Proceedings of the 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems*, pp. 173–180, ACM Press, 2013.

[5] A. Varga and others, “The OMNeT++ discrete event simulation system,” in *Proceedings of the European simulation multiconference (ESM2001)*, vol. 9, p. 65, sn, 2001.

[6] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. Haneveld, T. E. Parker, O. W. Visser, H. S. Lichte, and S. Valentin, “Simulating wireless and mobile networks in OMNeT++ the MiXiM vision,” in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, p. 71, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[7] A. Förster, “Implementation of the B-MAC Protocol for WSN MiXiM,” in *4th international workshop to be held in conjunction with Simutools*, 2011.

[8] IEEE Computer Society, LAN/MAN Standards Committee, Institute of Electrical and Electronics Engineers, and IEEE-SA Standards Board, *IEEE standard for local and metropolitan area networks. Part 15.4, Part 15.4,*. New York: Institute of Electrical and Electronics Engineers, 2011.

Looking into Hardware-in-the-Loop Coupling of OMNeT++ and RoSeNet

Sebastian Boehm and Michael Kirsche
Computer Networks and Communication Systems Group
Brandenburg University of Technology Cottbus-Senftenberg, Germany
eMail: {sebastian.boehm, michael.kirsche}@b-tu.de

***Abstract*—*Network emulation* using real sensor node hardware is used to increase the accuracy of pure network simulations. Coupling OMNeT++ with network emulation platforms and tools introduces new application possibilities for both sides. This work-in-progress report covers our experiences of using OMNeT++ as a test driver for RoSeNet, a network emulation and test platform for low-power wireless technologies like IEEE 802.15.4. OMNeT++ and RoSeNet were interconnected to enable a co-simulation of real sensor networks with a MAC layer simulation model. Experiences and insights on this Hardware-in-the-Loop (HIL) simulation together with ideas to extend OMNeT++ and to provide a generic interconnection API complete the report.**

***Index Terms*—Hardware-in-the-Loop, Co-Simulation, Network Emulation, Sensor Network Emulator, OMNeT++, RoSeNet**

I. Introduction

The evaluation of wireless networks requires accurate and extensive testing using tools and concepts of network testbeds such as packet, routing and latency analyses. Due to the underlying conditions in the wireless spectrum, a node's energy consumption and performance aspects also need to be considered. The performance and features of distributed sensor applications strongly depend on the underlying hardware resources and the channel conditions. This performance impact can usually only be tested and debugged on the actual sensor node hardware. With an increasing number of nodes in a network, installing and configuring application-specific testbeds is a time consuming and often expensive task.

Different crucial network evaluation techniques for Wireless Sensor Networks (WSNs) are presented below. We emphasize the necessity of network emulation in general and motivate our contribution of coupling OMNeT++-based network simulation with emulation tools like RoSeNet in context of WSNs.

A. Simulation of Wireless Sensor Networks

Within network simulations, it is relatively easy to define network topologies, layered architectures, and algorithmic applications. In the simulation flow, where system states change at discrete points over time, parameters can be customized individually to determine performance factors. In the context of WSNs however, simulations are limited [1] and lack realism. Simulators in general may not provide accurate modeling of real execution times at node level. Nodes are commonly simple and abstract entities, where actual hardware resources do not have any representation. Suitable radio channel and physical environment conditions are typically abstracted or omitted completely; both have to be considered when developing applications for resource constrained devices working in wireless environments. Wireless environments and channel conditions are depicted and abstracted by mathematical functions from varying complexity. There is a big tradeoff between realistic simulation models and their performance and scalability. A simple example is the comparison between the performance and realism of wireless channel models like the disk or free space model, the log normal shadowing model, and the multi-path ray tracing model. Their performance is decreasing while their correctness is increasing (although at different rates).

B. Wireless Sensor Network Testbeds

For the case of distributed sensor networks, it is usually complex and costly to manage physical testbeds with all required analysis features. Nodes within testbeds are not isolated from their surrounding radio environment, which is difficult to predict and measure. Common physical WSN testbeds lack controllability and easy configuration of the physical environment and are often limited or fixed in scale. In general such testbeds cannot enable easy adjustments of application or protocol parameters as it is common in network simulators. Applications run as static implementations on sensor nodes, which is why live debugging and detailed monitoring of system states and traces during the application execution are often not provided inside testbeds. Firmware extensions and extended runtime environments with monitoring capabilities are options to provide such information in testbeds. However, their maintenance is time-consuming and interventions in the running implementation can introduce unforeseen behavior and errors. Common WSN experimentation testbeds are presented and compared in [2].

C. Wireless Sensor Network Emulators

Controllable operational and environmental system conditions for physical signal transmissions are feasible in hardware-based network emulation. The network emulation concept forms an entire evaluation system through a combination of various real, virtual and abstract components. In the context of developing a sensor network application, the actual target hardware is still essential because both the PHY and (partially) the MAC layer are implemented in hardware and hardware constraints play an important role in application and protocol design. An interesting evaluation technique is provided by

wireless channel emulators. Practical deployments vary from laboratory test setups with coaxial-based radio links [3] to complex analog or digital radio channel emulators [4, 5]. In such test setups, real hardware nodes are shielded and connected over their radio interfaces to a hardware channel simulator, where effects of signal propagation are emulated. In the context of current wireless spectrum dynamics, it could be very important to test node networks and wireless applications under detailed wireless influences. Nevertheless, the traffic in such emulation and test setups will be created by sensor nodes running custom applications and static protocol implementations, comparable to testbeds.

To transcend the limitations of both, simulation and wireless emulation systems, our suggestion is to combine both techniques in a kind of co-simulation, which provides a runtime environment to conduct wireless network experiments under realistic channel conditions by using real WSN hardware including their transceivers. Early survey reports like [6] showed that there are open research questions regarding the integration of tools to support both simulation and emulation. We want to encourage interest in this topic and stimulate discussions with other OMNeT++ community members, to refine the coupling of real hardware with OMNeT++ in the context of WSN simulation.

This work-in-progress report covers our experiences using OMNeT++ in context of hardware-based network emulation with RoSeNet. It is structured as follows: In section II, we introduce RoSeNet, its emulation concept, and our current setup that uses OMNeT++ as a test driver. We also introduce an existing co-simulation approach and an accurate simulation model for IEEE 802.15.4, which is a common PHY / MAC standard in the domain of WSNs. While using OMNeT++ in our test case as the preliminary stage of coupling, section III presents our strategy for a seamless combination of OMNeT++ and RoSeNet on a protocol level. In section IV, important aspects and the main benefits of our contribution are presented. Section V closes this report with a short discussion about the benefits of a generic co-simulation interface for OMNeT++.

II. Background

Our motivation to combine network simulation with emulation tools derives from our main field of research (i.e., network emulation) and our previous work with the network emulation platform RoSeNet. This section covers (a) our setup of using OMNeT++ as a test driver, similar to the co-simulation strategy used by Veins [7], and (b) an accurate IEEE 802.15.4 simulation model [8], necessary for simulation-based network emulation from our point of view.

A. RoSeNet

RoSeNet [9] is a network emulation and test platform for low-power wireless technologies and WSNs, developed by dresden elektronik[1], which focuses on hardware-based channel emulation. The platform consists of individual panels with replaceable sensor node hardware, including custom application and protocol implementations. The overall architecture incorporates up to 1000 sensor nodes, which enables emulation of large-scale networks. All sensor nodes on the platform are interconnected through a controllable coaxial cable radio environment. RoSeNet can hence be described as a wireless channel emulator where signal propagation can be adjusted by digital step attenuation. Interference signals can be injected with the help of signal supply points. The status of RoSeNet can be considered as "in development".

During our work with the RoSeNet network emulator, we developed an interface that enables us to "feed" real sensor node hardware on the emulation panels with generated MAC layer protocol data. Developers can thus achieve a decent control over the communication flow in the network and construct various test scenarios. To generate traffic for the RoSeNet emulation platform, we inject frames using OMNeT++ as a test driver. We created an example in OMNeT++ / INET to feed our emulation setup with generated MAC protocol data units. For a test case, we simulated a scenario with TCP/IP traffic over Ethernet. We hence transmitted non-compliant protocol data to the destination hardware over the Hardware-in-the-Loop (HIL) interface in this "first step" scenario.

To overcome difficulties in creating external sensor node interfaces in OMNeT++, we used and extended INET's `PcapRecorder`. PCAP is a widely used standard for the exchange of communication data over program and system boundaries. Because PCAP frames already include real binary packet formats for most of the popular and currently used protocols, we have a common and widely used standard for exchanging our communication data. We implemented changes to utilize `PcapRecorder` together with a network dumper. Within the initialization, the `PcapRecorder` establishes a TCP connection with the `PcapDumper` module to the emulation server of the RoSeNet system. The parameters of the socket connection are set in the simulation configuration file (compare Listing 1).

Listing 1. Socket connection configuration - excerpt from `omnetpp.ini`

```
**.server.numPcapRec = 1
**.server.pcapRecorder[0].networkEnabled = true
**.server.pcapRecorder[0].serverIP = "localhost"
**.server.pcapRecorder[0].serverPort = 4242
**.server.pcapRecorder[0].pcapFile = "results/server.pcap"
```

Simulated packets can now be transmitted through the socket connection during simulation runtime. Inside RoSeNet's software system, these packets are interpreted, encapsulated, and forwarded to the specified nodes over their serial interface via the Serial Line Internet Protocol (SLIP). Incoming protocol data packets at node level are transmitted over the network stack. On the return path, radio packets will be forwarded to RoSeNet's software system, encapsulated in the PCAP format and transmitted to any registered receiver, even back to the simulation system (OMNeT++ in our case). The flow chart in Figure 1 only depicts a simplified forwarding of packet data from OMNeT++ via the sub-modules of the RoSeNet emulation system to the radio hardware.

[1]RoSeNet research project: https://www.dresden-elektronik.de/funktechnik/wireless/research-projects/rosenet/?L=1

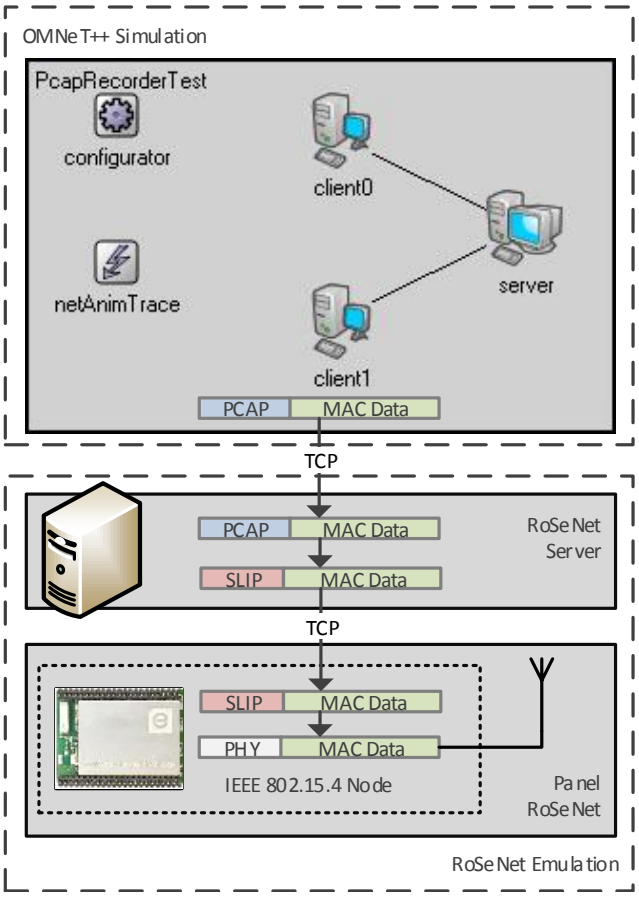


Fig. 1. Simplified message exchange between OMNeT++ and RoSeNet

B. Co-Simulation in OMNeT++

We motivated the need for a simulation of both physical and communication network dynamics of WSNs in Section I. A comparable concept for a co-simulation of OMNeT++ with another tool (software tool in this case) can be found in the area of mobility simulation, which involves highly dynamic and complex systems. Actor mobility is a major concern in the research field of Vehicular Ad Hoc Networks (VANETs). The Veins [7] framework uses a co-simulation concept for a bidirectional coupling of OMNeT++ and the SUMO (Simulation of Urban MObility) traffic simulator. Coupling and communication are controlled and synchronized over the Traffic Control Interface (TraCI) [10]. TraCI uses a TCP socket connection to enable a bidirectional exchange of command and response messages between both Veins (OMNeT++) and SUMO. The co-simulation concept of Veins and SUMO is based on the processing of internal and external events, whereupon OMNeT++ retains control over the simulation flow.

Retaining control of RoSeNet's software and runtime system for test scenarios through the use of OMNeT++ is also our motivation. We hence need an accurate representation of the IEEE 802.15.4 standard in OMNeT++ / INET, to exchange the necessary protocol primitives with the real hardware transceiver implementations.

C. IEEE 802.15.4 Simulation Model

We focused on a new IEEE 802.15.4 simulation model for OMNeT++ / INET that was previously introduced at the OMNeT++ Community Summit in 2014 [8]. The simulation model was built to emulate the complex behavior of the IEEE 802.15.4 standard in the 2006 revision. Protocol operations and service primitives were designed to be compliant with the standard. A detailed introduction of the simulation model is available in [8]. Figure 2 shows an IEEE 802.15.4 example host with a network stack consisting of different submodules.

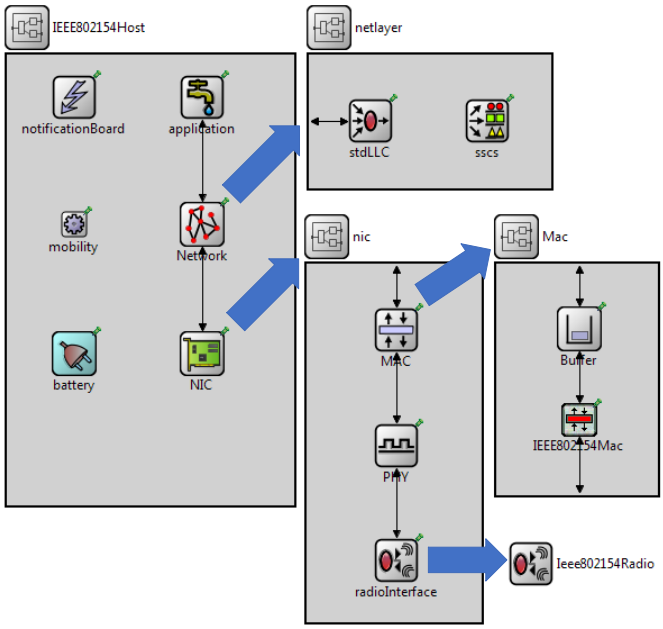


Fig. 2. An IEEE 802.15.4 host with its submodules (taken from [8])

III. Simulation-based Network Emulation

In this section, we introduce and motivate a strategy that combines model-based network simulations and real sensor hardware to increase the accuracy of WSN evaluation. Some parts in this combination strategy, particularly the higher layers of the protocol stack, are simulated, while other parts are represented by real hardware components interconnected through HIL interfaces. For our own work, we want to extend OMNeT++'s capabilities to enable physical channel emulation and hardware-related profiling of protocol performance characteristics in sensor networks through the exchange of protocol and control data units among OMNeT++ and RoSeNet. This concept of bridging between simulated and physical real nodes could speed up and expand the possibilities of wireless embedded system design. Other application areas would also benefit from such a coupling of OMNeT++ with external tools over a desired common interface.

We argue that the interconnection of network simulation and network emulation platforms and tools, where real physical wireless network and sensor interfaces are used, could be a valuable extension for an OMNeT++-based network analysis

and performance evaluation toolset. The exchange of real and simulated protocol data can enhance accuracy in a similar way that a preparation of simulations with physical sensor readings and protocol data could improve the exactness of simulations when compared to testbeds. Network emulators, on the other hand, could also be fed with simulated protocol data to verify and validate used simulation models. Such interfaces are still not included in the area of WSN simulation.

Our suggestion is a hybrid network emulation system in a HIL or emulation-assisted simulation environment, where the emulation setup is controlled by the network simulation. In practical terms, OMNeT++ exercises control over the sensor hardware or the hardware emulation system. To enable this, extensions of the simulation framework are planned to support parallel execution of protocol simulation and physical packet transmission. The proposed approach allows to include the physical details (e.g., radio transceiver or sensor interfaces) of a sensor node in protocol simulations, which are very costly to model but often feasible in testbeds or network emulation platforms like RoSeNet. To avoid being reliant on a specific node platform, we focused on a common interface that includes mechanisms for sending and receiving MAC packets as well as service primitives conform to the IEEE 802.15.4 standard over the boundaries of the simulation domain. Our contribution is also to provide OMNeT++ with control interfaces for acquiring data from different sensor node hardware or emulation systems.

A. HIL Interface Architecture

The resulting co-simulation interface is responsible for controlling interactions and the communication flow between simulated and real or emulated nodes. Developers could replace some simulated nodes within their scenarios with physical ones to verify hardware-depending features. Furthermore, we do not assume that both evaluation domains run on the same host system. The synchronization and control scheme between simulator and real sensor hardware could be realized similar to the Veins approach, focused on controlling the hardware runtime system by the simulation. Therefore, a number of components are necessary to generate emulation parameters, exchange messages in real packet format, and manage a synchronized real-time communication and control flow. To facilitate a consistent and universal communication interface solution, we focus on exchanging protocol data at the MAC layer using PCAP, like we first did while using OMNeT++ / INET as a simple test driver.

1) External Interfaces: In context of IEEE 802.15.4 and the used simulation model [8], the basic component in OMNeT++ / INET is represented as a protocol-specific external interface (e.g. `IEEE802154ExtInterface`) that handles communication data as events from both abstract simulation and emulation domain. A structural architecture of this external interface could be comparable to Irene Ruengeler's approach, where INET is connected with real networks [11, 12]. Figure 3 shows an example external interface for IEEE 802.15.4. Protocol-conform serializers and de-serializers will be used to convert simulated protocol data to the real binary packet format and back. Receiving and forwarding of packets needs to be implemented by a custom PCAP scheduler with access to the HIL interface. To enable the control of a node's real physical interface from within OMNeT++, we have to identify possible exchange points for the transfer of protocol messages between simulated and real layers. Those exchange points could be implemented in the node's firmware or its operating system.

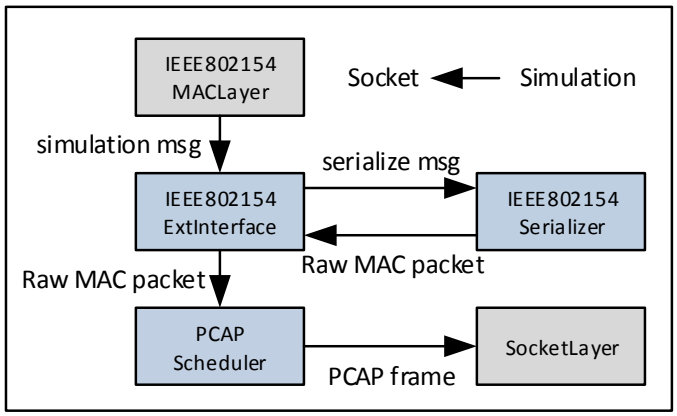


Fig. 3. Example IEEE 802.15.4 external interface in OMNeT++ / INET

2) Emulation Control Interface: The communication and control flow of the HIL approach is ensured over an Emulation Control Interface (EmuCI), similar to the TraCI approach. A connection to the emulator will be realized over a TCP socket in conjunction with custom schedulers that take responsibility for communication and control flows. The interface needs to provide all necessary mechanisms to control the emulator and all emulation-characteristic parameters. The inter-domain transmission of simulated and real packets can be realized with a communication-related PCAP scheduler. Parameters and settings should be included in the simulation configuration process to simplify things.

IV. Scenarios and Benefits

In the context of network evaluation in realistic environments, the objective is to couple the event-based network simulation with network emulation tools or real sensor node hardware through an interface that fulfills the requirements of a realistic evaluation using hardware-based emulation. This way, it is possible to control the behavior of emulated WSN environments to understand the influences of wireless channels and hardware characteristics from an abstract network simulation's point of view. During simulation runtime, adaptations to external real or emulated components could be enforced.

Important aspects for a validation of radio transmissions and wireless mediums are not integrated in available simulators for wireless sensor networks. A few desired benefits of the simulation-controlled emulation concept are:

- the ability to feed a WSN simulation with real application data and sensor readings,
- feed the sensor node hardware in context of implemented test applications with simulated traffic and network data,

- enable a power profiling of communication overhead at the PHY layer,
- wireless channel emulation support for simulation, e.g. injecting interferences and non-compliant protocol data or jamming the radio communication.

V. Discussion

These contributions shall motivate discussions about the benefits of a generic interface for co-simulation architectures. From our point of view, a generic co-simulation interface instead of an application- or framework-specific one is an interesting extension for OMNeT++. With Internet of Things scenarios, VANETs, intelligent sensor networks for Smart Grid or medical applications, and Cyber Physical Systems (CPSs) interacting with cloud services, there is a growing demand to simulate hardware or environment-related details and network protocol behavior simultaneously in a comprehensive manner. The exchange of system events on various levels is an objective. The event-driven concept behind network simulation with OMNeT++ would be preserved and extended with external events. Communication-specific details and protocol message flow should be allowed at different protocol layers.

VI. Final Remarks

We want to state that the presented approach is work-in-progress, thus still in a high-level state without fully laid out technical details. The conceptual design behind our ongoing research and the proposed ideas should be not limited to any specific emulation system, hardware resource, or network simulation environment. We will however focus on OMNeT++ and RoSeNet for the actual implementation and testing.

References

[1] Martin Stehlik. "Comparison of Simulators for Wireless Sensor Networks". M.Sc. Thesis. Faculty of Informatics, Masaryk University, 2011.

[2] A. K. Dwivedi and O. P. Vyas. "An Exploratory Study of Experimental Tools for Wireless Sensor Networks". In: *Wireless Sensor Network* 3.7 (2011), pp. 215–240.

[3] R. Burchfield et al. "RF in the Jungle: Effect of Environment Assumptions on Wireless Experiment Repeatability". In: *Communications, 2009. ICC '09. IEEE International Conference on*. June 2009, pp. 1–6. DOI: 10.1109/ICC.2009.5199421.

[4] Anite. *Propsim channel emulation solutions*. 2008. URL: http://www.anite.com/propsim.

[5] Kevin C. Borries et al. "FPGA-Based Channel Simulator for a Wireless Network Emulator." In: *VTC Spring*. IEEE, 2009. DOI: 10.1109/VETECS.2009.5073565.

[6] Muhammad Imran, Abas Md Said, and Halabi Hasbullah. "A Survey of Simulators, Emulators and Testbeds for Wireless Sensor Networks". In: *2010 International Symposium in Information Technology (ITSim)*. IEEE, 2010. DOI: 10.1109/ITSIM.2010.5561571.

[7] Christoph Sommer, Reinhard German, and Falko Dressler. "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis". In: *IEEE Transactions on Mobile Computing* 10.1 (Jan. 2011), pp. 3–15. DOI: 10.1109/TMC.2010.133.

[8] Michael Kirsche and Matti Schnurbusch. "A New IEEE 802.15.4 Simulation Model for OMNeT++ / INET". In: *Proceedings of the 1st OMNeT++ Community Summit (OMNeT 2014)*. Sept. 2014. URL: http://arxiv.org/abs/1409.1177.

[9] Michael Galetzka et al. *Verbundprojekt: Entwicklung von Methoden und Verfahren für den Aufbau von robusten und funktionssicheren drahtlosen Sensor-Aktor-Netzwerken*. Tech. rep. dresden elektronik Ingenieurtechnik GmbH, Fraunhofer-Institut für Integrierte Schaltungen, Germany, June 2012.

[10] Axel Wegener et al. "TraCI: An Interface for Coupling Road Traffic and Network Simulators". In: *Proceedings of the 11th Communications and Networking Simulation Symposium (CNS 2008)*. Ottawa, Canada: ACM, 2008, pp. 155–163. DOI: 10.1145/1400713.1400740.

[11] Irene Rüngeler. "SCTP - Evaluating, Improving and Extending the Protocol for Broader Deployment". Dissertation. University Duisburg-Essen, Dec. 2009.

[12] Michael Tüxen, Irene Rüngeler, and Erwin P. Rathgeb. "Interface Connecting the INET Simulation Framework with the Real World". In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques (SIMUTools 2008)*. Marseille, France: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, 40:1–40:6. DOI: 10.1145/1416222.1416267.

Implementation of PFC and RCM for RoCEv2 Simulation in OMNeT++

Qian Liu*, Robert D. Russell*, Fabrice Mizero†, Malathi Veeraraghavan†, John Dennis‡, Benjamin Jamroz‡
*Department of Computer Science, University of New Hampshire, Durham, NH, USA
†Dept. of Electrical & Computer Engineering, University of Virginia, Charlottesville, VA, USA
‡National Center for Atmospheric Research, Boulder, CO, USA
Email: *{qga2, rdr}@unh.edu †{fm9ab, mv5g}@virginia.edu ‡{dennis, jamroz}@ucar.edu

***Abstract*—As traffic patterns and network topologies become more and more complicated in current enterprise data centers and TOP500 supercomputers, the probability of network congestion increases. If no countermeasures are taken, network congestion causes long communication delays and degrades network performance. A congestion control mechanism is often provided to reduce the consequences of congestion. However, it is usually difficult to configure and activate a congestion control mechanism in production clusters and supercomputers due to concerns that it may negatively impact jobs if the mechanism is not appropriately configured. Therefore, simulations for these situations are necessary to identify congestion points and sources, and more importantly, to determine optimal settings that can be utilized to reduce congestion in those complicated networks. In this paper, we use OMNeT++ to implement the IEEE 802.1Qbb Priority-based Flow Control (PFC) and RoCEv2 Congestion Management (RCM) in order to simulate clusters with RoCEv2 interconnects.**

***Keywords*-Congestion Control; Flow Control; OMNeT++; RoCEv2**

I. Introduction

Most networks have congestion points or bottlenecks. Network congestion usually occurs when the total demand for a link is greater than the capacity of the link. In lossless networks such as InfiniBand [1] and RDMA over Converged Ethernet version 2 (RoCEv2) [2], the impact of congestion can be severe and can cause long communication delay.

In current enterprise data centers and TOP500 [3] supercomputers that employ InfiniBand, network congestion is inevitable as traffic patterns and network topology become complicated. Jobs running on supercomputers have experienced significant variation in latency due to congestion [4, 5]. However, huge clusters seldom utilize congestion control due to concerns that, if they are not appropriately configured, such mechanisms may negatively impact production jobs. Therefore, simulations for those congestion situations, especially situations in today's huge clusters that form supercomputers, are needed to identify congestion points and sources without modifying or rewriting the applications experiencing congestion, and more importantly, to determine optimal settings for the congestion control mechanism.

OMNeT++ [6] is an extensible, modular, component-based C++ network event simulator. Network components and basic elements can be organized in modules, which can be connected via communication gates. An open-source OMNeT++ simulation model [7] released by Mellanox implements the InfiniBand credit-based flow control mechanism and Quality-of-Service (QoS) that supports arbitration among different Virtual Lanes (VLs). Integration of InfiniBand congestion control into this model was discussed in [8], but that implementation is not yet open-source.

In this paper, we extend the current InfiniBand OMNeT++ model [7] released by Mellanox (abbr. the MLNX model) with Priority-based Flow Control (PFC, IEEE 802.1Qbb [9]) and RoCEv2 Congestion Management (RCM) [2] in order to simulate RoCEv2 clusters. The reason we based our work on the MLNX model is that latest Mellanox CAs with 2 ports can be configured to run both ports as InfiniBand, both as RoCE, or to run one port in each mode. Therefore, internally they must have a lot of shared hardware/firmware and be based on nearly identical architectures. Additionally, the MLNX simulation model avoids InfiniBand details by dealing abstractly with data arbitration, transmission and forwarding based on QoS, concepts that are very close to, and can be reused in, RoCEv2.

II. Background

The InfiniBand architecture [1] defines the 4 lowest layers of the OSI reference stack. RoCE [10] preserves InfiniBand's verbs interface and its transport and network layers, but utilizes Ethernet's link and physical layers as well as their management infrastructure. Its packets are not routable. RoCEv2 [2] preserves InfiniBand's verbs interface and transport layer, but utilizes standard IP layer and Ethernet's link and physical layers. RoCEv2 packets can be routed.

TABLE I: Flow/Congestion Control Mechanisms in InfiniBand, RoCE and RoCEv2. (O means "can be utilized", × means "cannot be utilized", √ means "defined in the spec")

	InfiniBand	RoCE	RoCEv2
IB CFC	√	×	×
PAUSE	×	O	O
PFC	×	O	O
IB CC	√	×	×
QCN	×	O	×
RCM	×	×	√

Table I compares the underlying flow control and congestion control mechanisms implemented in InfiniBand, RoCE and RoCEv2. Lossless behavior in InfiniBand is achieved by its Credit-based Flow Control (CFC) mechanism. While RoCE [10] doesn't specifically require losslessness, its performance suffers if link layer flow control

mechanism is not provided [11]. The RoCEv2 specification [2] doesn't define a mechanism to achieve losslessness but it requires such behavior from the network, link, and physical layers below its InfiniBand transport layer. Losslessness in RoCEv2 can be achieved through the use of a link-layer flow control mechanism such as the Priority-based Flow Control (PFC) defined in IEEE 802.1Qbb [9], which extends the IEEE 802.3x PAUSE semantics to apply to multiple classes of service (Virtual Links, VLs).

With the introduction of link-layer flow control, packets are not longer dropped due to buffer overflow. Instead, a packet is simply not sent on the link unless the other end of the link has buffer space to receive it. This may cause the sending side buffer to fill, so that congestion spreads upstream. The InfiniBand congestion control mechanism is defined in [1] as a way to reduce congestion and its spreading. The RoCE specification [10] doesn't define any congestion countermeasures. However, RoCEv2 [2] provides its own level 3 end-to-end "RoCEv2 Congestion Management" (RCM) mechanism [2] that seeks to alleviate congestion. Network Elements (NEs, routers or switches) and Channel Adapters (CAs, host interfaces) play different roles in this mechanism. NEs are responsible for detecting congestion and notifying destination end points. For congestion notification, RCM relies on the IP mechanism defined in Explicit Congestion Notification (ECN) [12], in which NEs mark exiting packets involved in congestion using the ECN field in the IP header. A CA that receives a packet with a value of '11' in its *IP.ECN* field is responsible for notifying the packet source about congestion by returning a Congestion Notification Packet (CNP). A CA that receives a CNP is responsible for reducing its packet injection rate.

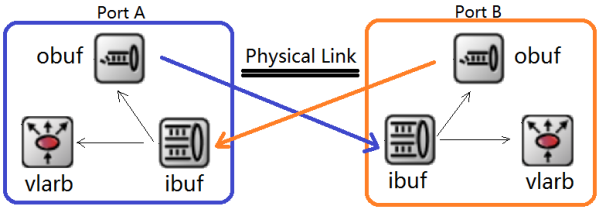


Figure. 1: Credit-based Flow Control (CFC) in the MLNX model

III. RoCE Priority-based Flow Control (PFC) Implementation

Figure 1 illustrates the credit-based flow control abstractions implemented in the current MLNX simulation model. In InfiniBand, one credit signifies 64 bytes of available receive buffer space. Whenever there is a change in credits on Port A, for instance, its *obuf* module sends a flow control packet across the physical link to the *ibuf* module in port B, which notifies its *vlarb* module about the number of available credits in port A that can be used to receive data packets from port B, and also notifies its *obuf* module about its local available credits, allowing the *obuf* module in port B to send a flow control packet to port A when necessary. Flow control packets should be sent frequently to prevent any possible long latency in data transmission. This frequency is configurable in the MLNX model.

Although the PFC mechanism is different from the credit-based flow control mechanism, the abstract architecture in Figure 1 can be reused in the implementation of PFC for RoCEv2. We observe that dual-ported Mellanox CAs can be configured to simultaneously run InfiniBand on one port and RoCEv2 on the other, implying that they must share a lot of common hardware/firmware. The *ibuf* and *obuf* modules are especially good building blocks for Ethernet NE simulation. We also continue to use the credit as a unit of available buffer space in the *ibuf* module. Because a PFC packet in [9] doesn't include available credit information, it doesn't have to be sent as frequently as in the credit-based flow control mechanism. Instead, it is sent when the buffer threshold on a port is exceeded in order to pause the other end of the VL early enough to prevent buffer overflow. More importantly, the PFC packet sending side must have enough buffer space available to store packets that might be in flight while the PFC packet is in transmission. There are two options to set a "high watermark" to trigger the PFC packet in our model, and users can select one of them. In the first option, our model uses the equation in [13] to calculate the watermark automatically. In the second option, users explicitly configure the value of this watermark.

According to [9], the PFC packet contains for each VL a pause duration whose value is based on a local estimate of when the buffer occupancy for that VL would be reduced. Further PFC packets may be necessary to refresh this duration if the situation persists. Since it is difficult to configure such an estimate, we implement the PFC pause duration as follows [13]. A large duration is specified in the PFC packet triggered by the "high watermark", and a "pause 0" packet is sent to resume traffic when triggered by a "low watermark" on that VL that is also configurable by users.

IV. RoCEv2 Congestion Management (RCM) Implementation

Traffic flow in the current MLNX model is illustrated in Figure 2. The *gen* module is responsible for generating packets by segmenting a message received from the upper level *app* into packets with a maximum sized payload that is configurable by the user. These generated packets are passed to the *vlarb* module for VL arbitration, after which they are sent out on the wire through the *obuf* module.

If a CA's *ibuf* module receives a packet, it forwards the packet to the *sink* module which consumes the packet. If a switch's *ibuf* module receives a packet, it forwards the packet to its requested output port's *vlarb* module, which arbitrates the packet and sends it out via the *obuf* module.

To add RCM into the MLNX model, the current module architecture and connections do not need to be modified, but several key components were added.

- **1. Congestion Detection in NEs**

In RCM, NEs such as switches mark packets upon congestion. But the RCM spec [2] doesn't define the

exact conditions when congestion should be detected. We implemented two kinds of congestion determination, and users can configure one for their simulation.

RCM-1a) We use the InfiniBand specification [1] definitions of the root of congestion and the victim of congestion, and the credit unit of 64 bytes for RoCEv2. Users can configure marking packets at the root alone or at both the root and the victim.

RCM-1b) We use a conventional definition of congestion, which defines the congestion state as the situation when the capacity of a requested output port is less than the sum of the traffic of the input ports competing for that output port, and packets built up in the input buffers exceed a fixed threshold value (configurable in our model). Packets will be marked at any congested points.

We monitor congestion in a NE port's *vlarb* module for each successful arbitration. If congestion is detected, the corresponding *obuf* module is set to the congestion state, which then causes it to begin to mark exiting packets.

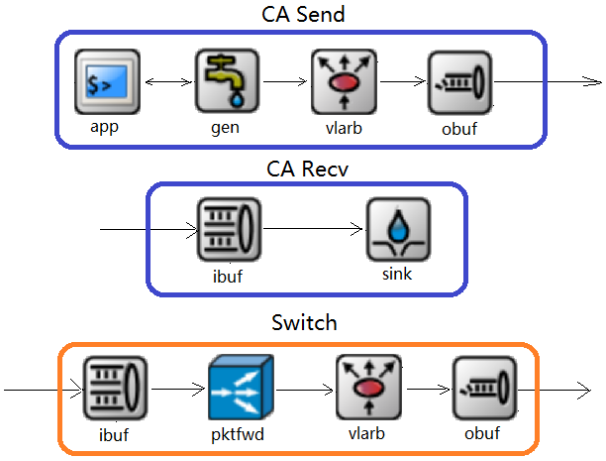


Figure. 2: Traffic Flow in the MLNX model

- **2. Packet Marking in NEs**

ECN marking simulation is done by setting a designated field in the current data packet structure (corresponding to the ECN field of the IP header).

- **3. Congestion Signaling in CAs**

When a CA's *sink* module receives a data packet with the *ECN* field set, it notifies its *gen* module which then generates a RoCEv2 CNP and sends it back to the source of the received packet via the *vlarb* module.

- **4. Injection Rate Reduction in CAs**

Upon reception of a CNP, the *sink* module notifies the *gen* module in its CA to reduce the injection rate of subsequent packets for the current connection on the specific VL. The amount of rate reduction is determined by parameters [2], but there is no guidance in the spec about how to set them. How the rate should be reduced can be configured by users. In our configuration for the example in section V we use a linear reduction strategy to implement the rate reduction. We configure T to be the time to transfer a data packet with length Maximum Transport Unit (MTU) on a wire. After receiving a CNP, the subsequent data packet won't be scheduled until $2\times T$ has passed; after receiving two CNPs, the subsequent data packet won't be scheduled until $3\times T$ has passed, and so forth.

- **5. Injection Rate Recovery in CAs**

A CA should increase its injection rate for a specific connection on a VL until a configurable amount of time has elapsed and/or a configurable number of bytes have been transmitted [2]. How the rate should be recovered can be configured by users. In our configuration for the example in section V each time a CA is able to recover the injection rate, it increases the rate back to the previous rate level. For instance, if the current injection rate is delayed by $3\times T$, the CA is able to increase the rate to a delay of only $2\times T$ for sending subsequent packets.

Both the time and the number of bytes in the recovery mechanism are also configurable in our simulator. It may require experimentation with the configuration to arrive at appropriate values that achieve optimal performance.

- **6. Congestion Statistics**

During simulation, information such as congestion duration and location are recorded. The sources and destinations of packets that flow through a congestion point are also recorded. In addition, various congestion related information such as packet latency and interval, the number of packets that are constrained by RCM and the constraint degree, etc, is collected as selected by user configuration.

V. Performance Evaluation

Figure 3 shows the simple topology used in our experiments. Four competing traffic flows from nodes A, B, C, and D send data simultaneously to node R. Packets are transferred on the same VL, and are placed onto the links as quickly as the links can accept them, subject to the PFC. All physical links are 40 Gbps, and the packet size is configured to 2048 bytes of data. Without congestion control, these competing traffic flows suffer from the Parking Lot unfairness problem [14].

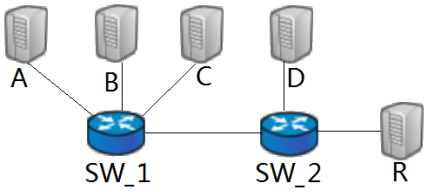


Figure. 3: Network with 4 sources simultaneously sending to dest R

We compare our simulation results with performance measurements obtained by running RoCE with PFC enabled on a hardware testbed having the topology shown in Figure 3. Switch 1 is a Mellanox SX1012, switch 2 is an Arista DCS-7050QX-32-R. Platforms A and B are equipped with Emulex Skyhawk CAs, one with firmware version 10.6.88.0, the other 10.6.144.10. Platform C is equipped with a MLNX MT27520 RoCE CA

with firmware version 2.33.5100. Platforms D and R have dual-ported MLNX MT27500 CAs with firmware version 2.32.5100 that supports both InfiniBand and RoCEv2. All platforms are running OFED 3.18 [15] on Scientific Linux 7.0 with kernel version 3.10.0. Figure 4 shows the throughput comparison for each sending node in both simulation and hardware test runs. Both performance curves are almost identical, within a difference of at most 3%, verifying the accuracy of our model.

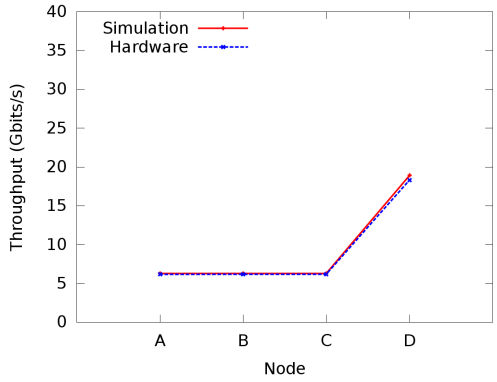


Figure. 4: Comparing simulation with RoCE hardware when PFC is enabled

Our simulations and hardware tests measure the user payload throughput observed at each node, which for a 40 Gbps link is a maximum of about 38 Gbps after accounting for protocol overhead. With PFC but without RCM[1], all input queues on both switches will fill quickly, at which point PFC will effectively reduce the injection rates of the sending nodes. Switch 2 will arbitrate equally its two input flows destined to node R, so that node D and the link from switch 1 will each see about 19 Gbps, half the maximum payload throughput of 38 Gbps seen at node R. Similarly, switch 1 will arbitrate equally its reduced output capacity of about 19 Gbps on its link to switch 2, so that flows from nodes A, B, and C are each reduced to about 6 Gbps, only one-sixth their maximum rated throughput.

Table II compares the simulated throughput of each flow in Figure 3 achieved in three scenarios: without RCM (NO RCM), RCM with the congestion detection mechanism in section IV-1a (RCM_1a), and RCM with the congestion detection mechanism in section IV-1b (RCM_1b). Without RCM, the throughput of D is three times that of each of A, B, and C due to Parking Lot unfairness, as already observed in Figure 4. With RCM_1a or RCM_1b, the four competing traffic flows share the link to receiver R almost equally. The results from RCM_1a and RCM_1b differ only slightly, indicating that these congestion detection mechanisms produce a similar effect with this simple network topology.

VI. Conclusion

In this work, we utilize OMNeT++ to simulate the IEEE 802.1Qbb Priority-based Flow Control (PFC), and RoCEv2 Congestion Management (RCM). Based on preliminary simulations, our implementation demonstrates that RCM reduces the negative effect of congestion in a simple network suffering the Parking Lot unfairness problem. Additionally, as all RoCEv2 parameters are configurable in our model, it can be easily customized and applied to any RoCEv2 cluster simulation.

[1]RCM is not available on all our hardware CAs

TABLE II: Simulated Throughput in Gbps in 3 RCM Scenarios

	NO RCM	RCM_1a	RCM_1b
A	6.3	9.37	9.29
B	6.3	9.42	9.35
C	6.3	9.51	9.43
D	18.9	9.72	9.69

Acknowledgment

The authors would like to thank our funding agencies. This research is supported in part by National Science Foundation grants OCI-1127228, OCI-1127340, and OCI-1127341.

References

[1] InfiniBand Trade Association, "Infiniband Architecture Specification Volume 1, Release 1.3," Mar. 2015.
[2] ——, "Supplement to InfiniBand Architecture Specification Volume 1 Release 1.2.1 Annex A17: RoCE-v2," Sep. 2014.
[3] "Top500 supercomputers." [Online]. Available: http://www.top500.org/
[4] A. Bhatele, K. Mohror, S. Langer, and K. Isaacs, "There Goes the Neighborhood: Performance Degradation due to Nearby Jobs," in *in SC'13*, Nov. 2013.
[5] A. Jokanovic, G. Rodriguez, J. Sancho, and J. Labarta, "Impact of Inter-application Contention in Current and Future HPC systems," in *18th IEEE Symposium on High Performance Interconnects*, Aug. 2010.
[6] "OMNeT++: Discrete Event Simulator." [Online]. Available: https://www.omnetpp.org/
[7] "OMNeT++ InfiniBand Flit Level Simulation Model." [Online]. Available: http://www.mellanox.com/page/omnet/
[8] E. G. Gran and S. Reinemo, "InfiniBand congestion control: modelling and validation," in *SIMUTools'11 Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, Mar. 2011.
[9] "IEEE 802.1Qbb: Priority-based Flow Control." [Online]. Available: http://www.ieee802.org/1/pages/802.1bb.html
[10] InfiniBand Trade Association, "Supplement to InfiniBand Architecture Specification Volume 1 Release 1.2.1 Annex A16: RDMA over Converged Ethernet (RoCE)," Apr. 2010.
[11] B. Tierney, E. Kissel, M. Swany, and E. Pouyoul, "Efficient data transfer protocols for big data," in *E-Science, 2012 IEEE 8th International Conference on*, Oct. 2012.
[12] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sep. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3168.txt
[13] "Priority flow control: Build reliable layer 2 infrastructure." [Online]. Available: http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9670/white_paper_c11-542809_ns783_Networking_Solutions_White_Paper.html
[14] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.
[15] "OpenFabrics Alliance." [Online]. Available: http://www.openfabrics.org

Regain Control of Growing Dependencies in OMNeT++ Simulations

Raphael Riebl
Technische Hochschule Ingolstadt
Research Centre
Email: raphael.riebl@thi.de

Christian Facchi
Technische Hochschule Ingolstadt
Research Centre
Email: christian.facchi@thi.de

***Abstract*—When designing simulation models, it is favourable to reuse existing models as far as possible to reduce the effort from the first idea to simulation results. Thanks to the OMNeT++ community, there are several toolboxes available covering a wide range of network communication protocols. However, it can be quite a daunting task to handle the build process when multiple existing simulation models need to be combined with custom sources. Project references provided by the OMNeT++ Integrated Development Environment (IDE) are just partly up to the task because it can be barely automated. For this reason, a new approach is presented to build complex simulation models with the help of CMake, which is a wide-spread build tool for C and C++ projects. The resulting toolchain allows to handle dependencies conveniently without need for any changes in upstream projects and also takes special care of OMNeT++ specific aspects.**

***Index Terms*—build system, CMake, dependency management, OMNeT++**

I. Introduction

The desire for using CMake [1] to build OMNeT++ [2] projects has its roots in a true story. When the authors started to use Veins [3] for their simulations, extensions were simply placed within the Veins project tree. Whereas some modifications were considered to be generally useful for Veins and submitted to the upstream project, a growing portion of sources evolved to form a project of its own, which is now called Artery [4]. A number of new modules — first scattered over the Veins source tree — eventually moved to a dedicated source directory, although still within the Veins project. Indeed, Artery does not only depend on Veins, but also on further libraries. Concretely, these are a purely C++ based implementation of Vehicle-to-X (V2X) protocols called Vanetza [5] as well as the quasi-standard Boost [6] libraries. Up to now, Artery managed these additional dependencies by extending Veins' *configure* script. For this purpose it reads a local, user-specific file containing the locations to Boost and Vanetza directories, which can vary from one system to another.

While this approach is working, further extensions to the *configure* script would let it become just more similar to CMake, which already provides proven and comprehensive mechanisms for dealing with differences between various build host environments and external dependencies. Unfortunately, current versions of OMNeT++ do not support CMake yet. Therefore, this paper presents various tools, which make CMake aware of OMNeT++'s Network Description (NED) folders and allow to import existing OMNeT++ projects as dependencies. We refer to these dependencies as "legacy projects" in the following. With a CMake based build system it is more convenient to integrate existing implementations of algorithms and suchlike, easing development of OMNeT++ simulation models. Since a simple merger of all dependencies into one project would make complex models unnecessarily huge and hard to keep up-to-date with constantly evolving third-party code, handling these dependencies externally alleviates development in the long term.

The current practice of building OMNeT++ projects is briefly analysed in Section II. Based on this analysis, the components of a joint OMNeT++/CMake build system are presented in Section III. These features are then demonstrated in Section IV using the example of Artery. We conclude with a summary of open challenges and stumbling blocks. Furthermore, suggestions for mainstream CMake support by OMNeT++ are given in Section V.

II. Analysis

While OMNeT++ itself relies on traditional Makefiles for building its libraries from sources, it provides the Makefile generator *opp_makemake* for creating Makefiles for OMNeT++ projects. Projects like Veins and INET [7] are based upon *opp_makemake*, although they have extended it by individual Python scripts to deal with project specific details. In the case of INET, this script is called *inet_featuretool* and is used for feeding configuration parameters to *opp_makemake* by extracting build information from INET's *.oppfeatures*, *.oppfeaturestate* and *.nedfolders* configuration files. The purpose of this process is to enable or disable INET features. Veins encapsulates the *opp_makemake* invocation in its *configure* script. By default, Veins is built as a shared library and run via OMNeT++'s *opp_run* application. The *configure* script defines the directories which shall be passed to *opp_run* for locating the built shared library and accompanying NED folders. As an option, the location of the INET framework can be given to the script as an argument, so INET modules can be used within a Veins simulation.

In order to facilitate adaptation of existing models and libraries with CMake, no changes to these legacy projects

should be required. Albeit their sources are generally available, maintaining changes for CMake compatibility alongside the upstream repositories is an unpleasant task. Furthermore, a generic solution excels individual CMake adaptation code for each legacy project. Thus, additional configuration and maintenance efforts are kept at a bare minimum or avoided completely beforehand.

III. Build system design

There exist three types of dependencies a CMake build system for OMNeT++ projects needs to be aware of. First of all, there are executables, such as the essential C++ compiler as well as *nedtool* (formely *opp_msgc*) bundled with OMNeT++, which is responsible for generating C++ sources and headers from OMNeT++ message descriptions. Secondly, there are static and shared libraries, which can be either plain libraries like Boost or those containing OMNeT++ simulation model code, e.g. the INET shared library. Last but not least, running simulations depends on configuration files like *omnetpp.ini* and NED files accompanying OMNeT++ modules defining their inter-connections and configuration parameters.

A. CMake fundamentals

A CMake project comprises a platform independent *CMakeLists.txt* file describing the required steps to build targets like executables and libraries. These descriptions make use of commands, variables and properties attached to e.g. targets, directories and source files. One of CMake's cornerstones is for sure its *find_package* mechanism, which supports the integration of a plethora of external dependencies out of the box. The build process of CMake projects involves three phases:

1) *Configuring* the build directory where host-specific information such as locations of dependency libraries and build-specific options like compiler flags are stored. This build configuration can be modified interactively, e.g. with *cmake-gui*.
2) *Generating* out of these information the required files for a native build tool, e.g. a Makefile or IDE project files. In contrast to *CMakeLists.txt*, these files are specific to the build directory and system they are generated for.
3) *Building* the project within the build directory by invoking the native build tool, e.g. GNU Make, or loading the generated build files into the appropriate IDE and launching the build there.

CMake can be extended in various ways, e.g. by creating macros defining custom functionalities building upon CMake's commands. For more details we point to the extensive CMake documentation [1].

All of our subsequently presented CMake extensions and code contributions are available online[1].

[1]https://github.com/riebl/artery/releases/tag/opp-summit2015

B. OMNeT++ simulation environment

Obviously, finding the OMNeT++ environment on the local host is a basic prerequisite for any OMNeT++ project. Unfortunately, neither CMake nor OMNeT++ are aware of each other yet, so there is no official support for CMake's aforementioned *find_package* command to integrate OMNeT++ in the build process. Therefore, we sustain *find_package* by providing the *FindOmnetPP.cmake* script file. It contains heuristics utilised by *find_package* to look for an OMNeT++ installation with its header include directory as well as libraries. If the *omnetpp* executable (the IDE) is within the host's PATH environment variable, the script is capable of detecting the OMNeT++ root directory automatically. OMNeT++ version information as well as several compile definitions are extracted from the therein located *Makefile.inc*. OMNeT++'s libraries are afterwards available as imported targets with appropriate include directories and compile definitions set as interface dependencies, i.e. those information are automatically added to any user target depending on an imported OMNeT++ target library. In contrast to a simple variable pointing at an OMNeT++ library, the employed target mechanism allows to differentiate between release and debug builds, so OMNeT++ libraries with the "d" suffix in their filename are used when a project is built in a debug configuration. Although the simulation model IPv6Suite [8] has used CMake already 10 years ago, the techniques used back then are not on a par with the facilities offered by recent versions of CMake, as e.g. the import of targets inclusive of accompanying target properties such as required header include directories.

C. Legacy OMNeT++ projects

While these extensions to the *find_package* mechanism are sufficient for building stand-alone OMNeT++ projects with CMake, referencing already existing projects requires further steps. Though adding CMake build files to such legacy projects is feasible, a fast adoption is unlikely and maintenance of two build systems – opp_makemake and CMake – can be a burden. Therefore, a technique to integrate existing OMNeT++ projects like Veins or INET without requiring changes in their respective code bases has a better chance of acceptance.

Fortunately, Makefiles generated by *opp_makemake* comprise the original arguments passed during the generation process, i.e. include directories, compile definitions as well as name, type and location of the produced project binary. These information are processed by the new *opp_cmake* tool, which generates a CMake file wrapping the legacy project's binary and build instructions required for using this legacy binary. The basic process is outlined in Figure 1. At first the legacy project, which is going to be used as a dependency later on, is built in the common way. Usually, a custom Makefile invokes *opp_makemake* with the correct set of arguments. The resulting Makefile includes rules to build the actual legacy binary, which is e.g. a shared library. During the generation phase of CMake, the new *opp_cmake* tool is executed, which parses the original arguments passed to *opp_makemake* from the Makefile and creates a CMake file with import targets for

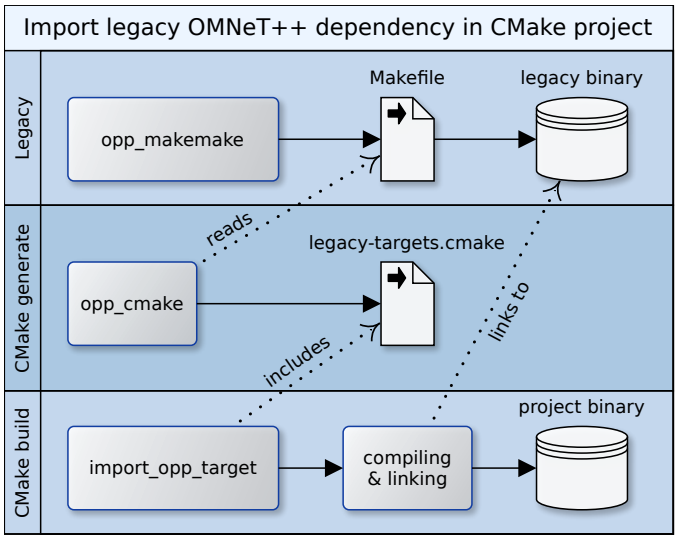


Fig. 1. Import of a legacy project as dependency of a CMake project

the legacy binary. Finally, this import target file is used via *import_opp_target* for the actual build phase, i.e. the CMake project binary can refer to the legacy binary as an ordinary CMake target. Thus, the project binary can be linked to the legacy binary easily.

It has to be noted, that a CMake project is only required to call the provided *import_opp_target* macro once. As part of that, the location to the Makefile generated by *opp_makemake* and the name of the import target are passed to the macro. Executing the *opp_cmake* tool as well as loading the import targets are handled by *import_opp_target*. Thus, this mechanism is very easy to use.

So far, *opp_cmake* is not capable to produce CMake targets for legacy projects using "deep includes", i.e. where all project directories are explicitly added to the compiler's header search path. This is not due to a general limitation, but the authors experienced no need for this feature up to now. Future versions of OMNeT++'s *opp_makemake* could also generate a CMake file with import targets alongside the present Makefile, which would make an additional tool like *opp_cmake* obsolete.

D. NED folders

A notable addition to OMNeT++ dependencies over conventional C/C++ dependencies are the NED folders containing network description files for each OMNeT++ *Simple Module*. These description files wire up the generic OMNeT++ runtime environment with user-provided C++ code and allow e.g. to set parameters of *Simple Module* objects through the textual simulation configuration. These NED files are not strictly required for the build process itself, but when the compiled simulation is to be executed with the help of *opp_run*. Besides the shared libraries containing the compiled *Simple Modules* it expects also the path to the accompanying NED folders. To account for this runtime requirement, imported targets are attributed with a novel NED_FOLDERS target property, which

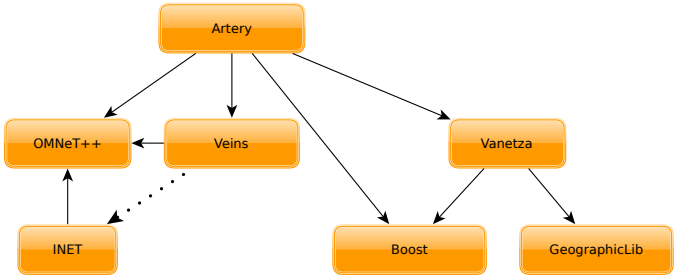


Fig. 2. Dependency graph of Artery

conveys the list of all NED folders specified in the legacy project's *.nedfolders* file.

By means of the *get_ned_folders* macro, all NED folders of a target and its dependencies can be aggregated recursively. Consequently, no NED folder is missed because every dependent folder is transitively added to the target. This feature enables the creation of run targets within CMake, which start simulations by means of *opp_run*, as the *add_opp_run* macro demonstrates. Alternatively, CMake could also create shell scripts including the NED folders information. CMake's *configure_file* command is predestined for such tasks.

IV. Demonstration

Practicability of the techniques presented in Section III is demonstrated using the example of the Artery project. The main components involved in this setup and their dependency relations are depicted in Figure 2. Quite obviously, components with custom *Simple Modules* depend on OMNeT++ base classes, just like Artery, INET and Veins do. INET is as of today not strictly required, but might be included implicitly through Veins' *configure* script. Artery, however, might incorporate INET through *import_opp_target* in the future as well, just as it does with Veins too. Additionally, Artery depends on the pure C++ library Vanetza. Since Vanetza's build system is already based on CMake, it can export its targets in its build tree. This works similar to the target import through *import_opp_target*, but here the dependency Vanetza is providing the CMake target file, whereas *opp_cmake* needs to be called by the dependant project. The Boost libraries and headers are integrated via the *find_package* module bundled with CMake. This mechanism is also used for Vanetza's GeographicLib dependency. Consumers of Vanetza, however, do not need to be aware of this dependency because it is transitively added to their dependencies when they just depend on Vanetza. Invoking the "run_example" target generated by CMake starts up the simulation model with an example configuration. If deemed necessary by the build system, changed sources of Artery are automatically rebuilt beforehand.

V. Conclusion

Thanks to the presented tools, it is now possible to build an OMNeT++ project using CMake and even incorporate further OMNeT++ projects as dependencies without changing their upstream code base. Despite the advances in handling

OMNeT++ projects with CMake, however, users still need to take care of installing appropriate versions of all dependencies. This can become slightly challenging in the presence of cross-dependencies, i.e. when multiple components depend on a common third-party library. Mixing several versions of one library into the final library should be avoided. When Veins is built with enabled INET support, other parts, e.g. Artery, should be built with the same INET version to avoid conflicts in the final executable. Same care should be taken regarding other dependencies like Boost.

Oddly enough, shared libraries intended to be executed via *opp_run* are still linked with OMNeT++ libraries. Experiments confirmed this is not required technically. Possibly, this could lead to problems when the shared library is linked with OMNeT++ libraries of a *release* build, whereas *opp_run* refers to *debug* libraries. It needs further investigation if there can arise problems due to this conjuncture.

Instead of creating CMake targets by reverse engineering with *opp_cmake*, *opp_makemake* could create a CMake target file next to the anyway generated Makefile without much effort. This approach would not even interfere with present mechanisms. Since *opp_makemake* is part of OMNeT++, no changes would need to be done to existing projects and therefore enable a smooth transition towards CMake.

References

[1] Kitware Inc. *et al.* (Jun. 9, 2015). CMake, [Online]. Available: http://www.cmake.org/documentation.

[2] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Simutools '08: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & workshops*, ICST, Brussels, Belgium, 2008, pp. 1–10.

[3] 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.

[4] R. Riebl, H.-J. Günther, C. Facchi, and L. Wolf, "Artery – extending Veins for VANET applications," in *Proceedings of the Fourth International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS 2015)*, Budapest, Hungary, Jun. 2015.

[5] R. Riebl *et al.* (Jun. 9, 2015). Vanetza, [Online]. Available: https://github.com/riebl/vanetza.

[6] B. Dawes, D. Abrahams, R. Rivera, *et al.* (Jun. 9, 2015). Boost C++ libraries, [Online]. Available: http://www.boost.org.

[7] A. Varga *et al.* (Feb. 2015). INET framework, [Online]. Available: http://inet.omnetpp.org (visited on 02/01/2015).

[8] J. Lai, E. Wu, A. Varga, Y. A. Şekercioğlu, and G. K. Egan, "A simulation suite for accurate modeling of IPv6 protocols," in *Proceedings of the 2nd International OMNeT++ Workshop*, Berlin, Germany, Jan. 2002.

Optimization in the Loop: Implementing and Testing Scheduling Algorithms with SimuLTE

Tutorial

Antonio Virdis
Dipartimento di Ingegneria dell'Informazione
University of Pisa
Pisa, Italy
a.virdis@iet.unipi.it

***Abstract*—One of the main purposes of discrete event simulators such as OMNeT++ is to test new algorithms or protocols in realistic environments. These often need to be benchmarked against optimal/theoretical results obtained by running commercial optimization solvers. The usual way to do this is to have the simulator run in a standalone mode and generate (few) *snapshots*, which are then fed to the optimization solvers. This allows one to compare the optimal and suboptimal solutions *in the snapshots*, but does not allow to assess how the system being studied would *evolve over time* if the optimal solution was enforced every time. This requires optimization software to run directly in the loop of the simulation, exchanging information with the latter. The goal of this tutorial is to show how to integrate a commercial solver (CPLEX) into the simulation loop of the OMNeT++ environment. For this purpose, we propose two methods: a first one that uses a solver as an external program, and a second one that exploits a C-written API for CPLEX known as *Callable Library*. We then exemplify how to apply these two methods to SimuLTE, a simulation model for LTE cellular networks, implementing and testing a simple solution to a well-known resource-scheduling problem.**

Keywords—OMNeT++; SimuLTE; resource allocation; optimization; LTE; CPLEX

I. INTRODUCTION

Event-driven simulators such as OMNeT++ are widely used by researchers to test and evaluate complex communication systems. They generally aim at either evaluating the performance of a known system or testing new algorithms on the system itself. This becomes even more useful when a new algorithm has to be compared against other solutions in order to evaluate a performance improvement. It is a common approach in this case to use optimal/theoretical results as terms of comparison: the problem is formulated as a mathematical model and the optimal solution is obtained using commercial solvers. The obtained results are then used to assess how close the new algorithm is to the theoretical best value, for example by computing the optimality ratio of some metrics.

In order to carry out the above method, we need a means to feed the optimization problem with actual inputs from the simulated system. This is usually done by running simulation campaigns using the algorithm that has to be tested and generating *snapshots* of the system status, i.e. collections of all the variables of the system that are relevant to the problem being considered. Each snapshot is then used to create an instance of the optimization problem that is then solved *offline*, using a commercial solver as a standalone program. As we can see in the left part of Fig. 1, the evolution of the simulated system depends only on the results produced by the heuristic algorithm under test, whereas the output of the solver is used only for performance comparison. In other words the optimal solution generated by the solver is never fed back to the simulated system, thus the solver is actually *outside* the simulation loop. This method requires only few modification to the code, and is effective when evaluating single snapshots. In general it does not show the long-term effects of using the optimal solutions as the system evolves.

A possible alternative to this method is shown in the right part of Fig. 1: two instances of the system are created with the same initial conditions. The two instances evolve independently, one taking feedbacks only from the heuristic, the other one only from the solver. The results of both systems are finally compared, allowing us to analyze the steady-state performance of the system.

The purpose of this tutorial is to describe methods to integrate optimization solvers into the simulation loop. We will propose two techniques and highlight their pros and cons. Then we will apply these methods to an INET-based simulation module for LTE networks, namely SimuLTE [1][2], using a well-known problem as an example. In the rest of this paper, we will first describe the outline of the tutorial in Section II, then conclusions will be draw in Section III.

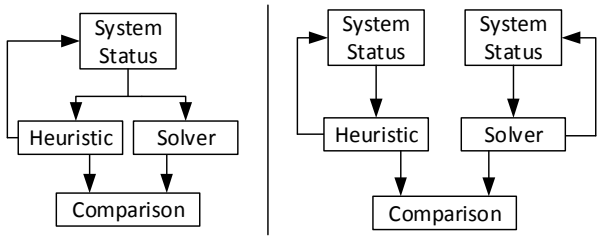


Fig. 1. Two examples of optimization usage in simulation: without feedback (left) and in the loop (right)

II. Outline of the Tutorial

The tutorial will be divided into two parts, one detailing the two integration methods, and the other exemplifying them with a practical example. A brief description of these parts follows.

A. Integrating optimization in the loop

First of all we describe two methods for integrating optimization into the simulation loop. We will use CPLEX [3] as an example of optimization software. The latter is a commercial product developed by IBM, made available to researchers via the IBM Academic Initiative. With reference to Fig. 2, we will consider a system composed of a Simulator (e.g. one based on OMNeT++) and an Optimization Tool. The former can be further split logically into one part implementing the system layers and another where the algorithm under study operates. The communication between the simulator and the Optimization Tool takes place whenever the algorithm is run and can be performed in four steps:

1. The allocation module reads all the relevant info from the system layers;
2. The allocation module builds an instance of the optimization problem and sends it to the optimization tool;
3. The optimization tool solves the problem and sends the solution back to the allocation module;
4. The allocation module parses the solution and enforces the algorithm decisions.

From a higher point of view we can say that steps 1 and 2 are responsible for feeding the optimization tool with the status of the system, whereas steps 3 and 4 implements the feedback operation.

The two integration methods we propose share the same general structure, but differ in the way the communication between the allocation module (i.e. the simulation environment) and the optimization tool is realized. The main idea behind the first method is to use CPLEX as an external program that is called during the simulation process. The tool being considered in this case is the CPLEX *Interactive Optimizer*, a command line interface to CPLEX. The communication between the two entities is made via file exchange, using a problem file in the LP format for step 2, and a solution file in XML format for step 3.

The second method we propose uses instead the CPLEX *Callable Library*. The latter is a set of C-written APIs that can be used to call CPLEX functions directly from the simulator code. For this purpose we first need to include the callable library into the OMNeT++ project. This way we can implement the communication in step 2 and 3 using simple C function calls. Moreover we will present a custom C++ interface for the *Callable Library* that can be used to simplify the writing of the problem.

We will discuss the pros and cons of the above methods, focusing on the trade-off between complexity of usage and performance (i.e. execution time). On one hand using the first method requires one only to write a file in LP format, which is rather simple, and to know how to parse an XML file.

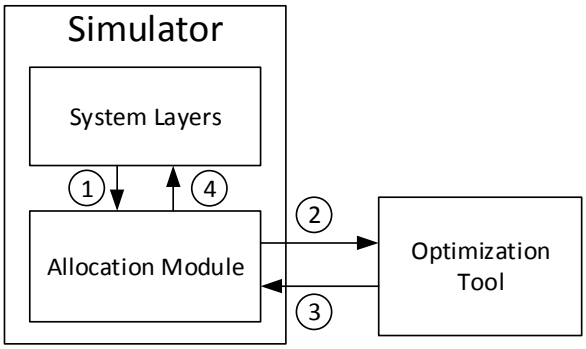


Fig. 2. Connection between a simulation module and an optimization tool

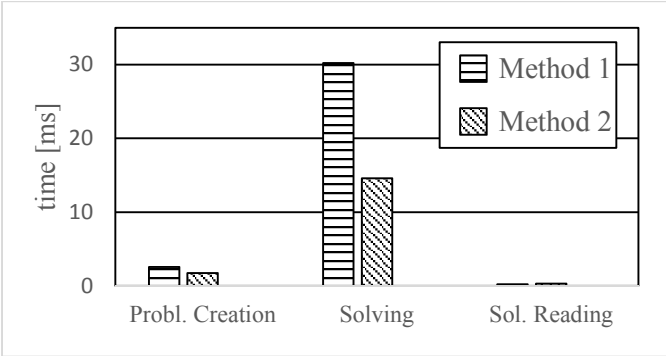


Fig. 3. Example of performance analysis of the two integration methods.

The second method requires instead knowledge of the Callable Library and its data structures. However, we will try to reduce the impact of this second issue with a custom C++ interface, as stated above. On the other hand, the first method is less efficient from the perspective of execution time. In fact it entails calling an external program, which requires frequent context switches, i.e. time consuming operations. The Callable Library is instead executed into the same context with a significant performance improvement. We will show how to measure the execution time of the two above methods, and how to evaluate the impact of each step of the process.

B. Scheduling algorithms with SimuLTE

In the second part of the tutorial we apply the two above methods to a practical problem using the *multiband scheduling problem* (MBS) as an example. The latter is a resource allocation problem typical of OFDM systems. We consider it in the in the context of a specific cellular technology, namely LTE. For this purpose we will use SimuLTE, an INET-based simulation model, focusing on its resource scheduling capabilities. We start by briefly describing the LTE system, with particular emphasis on resource allocation. For each characteristic of the system we give an explanation of the modeling assumption of the corresponding element in SimuLTE.

We then discuss the idea behind the MBS problem and give detail on how we can solve it both at optimality and heuristically. We show how to create the optimization problem and how to implement both integration methods into SimuLTE. For this purpose we divide this process into three phases that are common to both methods:

- a *problem creation* phase where the optimization problem is built;
- a *solving* phase during which the optimization tool is computing the solution;
- and a *solution reading* phase where the solution is read and stored into the simulator environment.

Finally we analyze the performance of the two integration methods from the point of view of execution time. This will help us to highlight the differences between them in order to better understand their pros and cons. An example of this analysis is shown in Fig. 3 where the execution time of each phase is compared.

III. Conclusions

The goal of this tutorial is to analyze the process of integrating optimization solvers into the simulation loop of OMNeT++. The tutorial is organized in two parts. We start by proposing two integration methods: a first one that uses CPLEX as an external program and a second one that calls it directly within the simulator code. We analyze pros and cons of both methods, giving particular emphasis. Then in the second part we consider a practical example describing and solving the problem of multiband scheduling in the context of LTE networks. In order to simulate the latter we use SimuLTE, an INET based module, and we show how to implement the presented methods into it.

References

[1] A. Virdis, G. Stea, G. Nardini, "SimuLTE: A Modular System-level Simulator for LTE/LTE-A Networks based on OMNeT++", SimulTech 2014, Vienna, AT, August 28-30, 2014

[2] SimLTE Website, http://simulte.com (accessed June 2015)

[3] IBM CPLEX Website, http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud (accessed June 2015)

A Tutorial of the Mobile Multimedia Wireless Sensor Network OMNeT++ Framework

Zhongliang Zhao*, Denis Rosário*,†, Torsten Braun*, Eduardo Cerqueira†
*Institute of Computer Science and Applied Mathematics, University of Bern, Switzerland
†Faculty of Computer Engineering and Telecommunication, Federal University of Para, Brazil
Email:zhao@iam.unibe.ch, denis@ufpa.br, braun@iam.unibe.ch, cerqueira@ufpa.br

***Abstract*—In this work, we will give a detailed tutorial instruction about how to use the Mobile Multi-Media Wireless Sensor Networks (M3WSN) simulation framework. The M3WSN framework has been published as a scientific paper in the 6th International Workshop on OMNeT++ (2013) [1]. M3WSN framework enables the multimedia transmission of real video sequence. Therefore, a set of multimedia algorithms, protocols, and services can be evaluated by using QoE metrics. Moreover, key video-related information, such as frame types, GoP length and intra-frame dependency can be used for creating new assessment and optimization solutions. To support mobility, M3WSN utilizes different mobility traces to enable the understanding of how the network behaves under mobile situations. This tutorial will cover how to install and configure the M3WSN framework, setting and running the experiments, creating mobility and video traces, and how to evaluate the performance of different protocols. The tutorial will be given in an environment of Ubuntu 12.04 LTS and OMNeT++ 4.2.**

***Index Terms*—Mobile Multimedia Wireless Sensor Networks, Simulation framework.**

I. Introduction

The rapid development of low-cost technologies involving camera sensors and scalar sensors have made Wireless Multimedia Sensor Networks (WMSNs) emerging topics. WMSNs promise a wide range of applications in Internet of Things (IoT) and Smart cities, such as environment surveillance, traffic monitoring, etc.

Many OMNeT++ frameworks have been proposed to study protocols in wired and wireless networks, in which Castalia includes advanced wireless channel and radio models, power consumption models, as well as MAC and routing protocols for wireless sensor networks (WSNs). However, Castalia does not provide any functionality for video transmission, control and evaluation as expected for emerging multimedia applications. This is mainly due to the fact Castalia was designed for scalar sensor network simulation.

Wireless Simulation Environment for Multimedia Networks (WiSE-MNet) [2] incorporates some of Castalias functionalities/features to provide a generic network-oriented simulation environment for WMSNs. WiSE-MNet addresses the need for co-designing network protocols and distributed algorithms for WMSNs. Even though designed for WMSNs, WiSE-MNet does not provide video control and QoE support, which is a key characteristic to enable multimedia evaluation from the users perspective. Additionally, it considers an idealistic communication mechanism to test algorithms without taking into account the unreliable nature of wireless medium. Moreover, WiSE-MNet does not support node mobility with complex traces as expected in many smart cities applications.

The Wireless Video Sensor Network (WVSN) model proposes a simulation model for video sensor networks. It defines the sensing range of camera nodes by a Field of View (FoV), which is more realistic for WMSNs. Additionally, depending on the number of nodes, the model determines the cover-sets for each sensor node and computes the percentage of coverage for each cover-set. Then, this information is used to increase the frame capture rate, e.g., a node has a higher capture rate when it has more covers. However, this work also fails in providing an accuracy video transmission and evaluation approach, and no mobility is supported.

In this context, it is clear that the existing OMNeT++ frameworks have no support for transmission, control and evaluation of real video sequences as required for many WMSNs and smart cities scenarios. Therefore, a QoE-aware and video-related framework that manages video flows with different characteristics, types, GoP lengths, and coding, are required. This framework should also be able to collect information about the type of every received/lost frame, frame delay, jitter and decoding errors, as well as inter and intra-frame dependency of the received/distorted videos, such that a set of mobile multimedia-based protocols can be evaluated and improved. M3WSN extends Castalia by integrating the functionalities of both WiSE-MNet and WVSN models, such that it supports transmission, control and evaluation of real video sequences in mobile WMSNs.

II. Getting Started

To support mobile object detection and tracking, certain image/video processing libraries are required. The Open Source Computer Vision Library (OpenCV) [3] is the most used library to detect, track, and understand the surrounding world captured by image sensors. OpenCV is released under a BSD license and hence it is free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports different operating systems. The library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform. Therefore, the first step is to install and configure OpenCV library.

III. OpenCV Installation

A. Installation Steps

To install and configure OpenCV, complete the following steps. The commands shown in each step can be copied and pasted directly into a Linux command line.

1. Remove any installed version of ffmpeg and x264:

```
$ sudo apt-get remove ffmpeg x264 libx264-dev
```

2. Get all the dependencies for x264 and ffmpeg:

```
$ sudo apt-get update
$ sudo apt-get install build-essential
    checkinstall git cmake libfaac-dev libjack-
    jackd2-dev libmp3lam-dev libopencore-amrnb-dev
     libopencore-amrwb-dev libsdl1.2-dev libtheora
    -dev libva-dev libvdpau-dev libvorbis-dev
    libx11-dev libsfixes-dev libxvidcore-dev
    texi2html yasm zlib1g-dev
```

3. Download and install gstreamer:

```
$ sudo apt-get install libgstreamer0.10-0
    libgstreamer0.10-dev gstreamer0.10-tools
    gstreamer0.10-plugins-base libgstreamer-
    plugins-base0.10-dev gstreamer0.10-plugins-
    good gstreamer0.10-plugins-ugly gstreamer0.10-
    plugins-bad gstreamer0.10-ffmpeg
```

4. Download and install gtk:

```
$ sudo apt-get install libgtk2.0-0 libgtk2.0-dev
```

5. Download and install libjpeg:

```
$ sudo apt-get install libjpeg8 libjpeg8-dev
```

6. Download, install, configure, and build x264 libraries:

```
$ wget ftp://ftp.videolan.org/pub/videolan/x264/
    snapshots/x264-snapshot-20120528-2245-stable.
    tar.bz2
$ tar xvf x264-snapshot-20120528-2245-stable.tar.
    bz2
$ cd x264-snapshot-20120528-2245-stable.tar.bz2
$ ./configure --enable-static
$ make
$ sudo make install
```

7. Download, install, and configure ffmpeg libraries:

```
$ wget http://ffmpeg.org/releases/ffmpeg-0.11.1.
    tar.bz2
$ tar xvf ffmpeg-0.11.1.tar.bz2
$ cd ffmpeg-0.11.1
$ ./configure --enable-gpl --enable-libfaac --
    enable-libmp3lame --enable-libopencore-amrnb
    --enable-libtheora --enable-libvorbis
$ make
$ sudo make install
```

8. Download and install v4l (video for Linux):

```
$ wget http://www.linuxtv.org/downloads/v4l-utils/
    v4l-utils-0.8.8.tar.bz2
$ tar xvf v4l-utils-0.8.8.tar.bz2
$ cd v4l-utils-0.8.8
$ make
$ sudo make install
```

9. Download and install OpenCV libraries:

```
$ wget http://downloads.sourceforge.net/project/
    opencvlibrary/opencv-unix/2.4.2/OpenCV-2.4.2.
    tar.bz2
$ tar xvf OpenCV-2.4.2.tar.bz2
$ cd OpenCV-2.4.2
$ make build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE
```

10. Configure Linux:

```
$ export LD_LIBRARY_PATH=/usr/local/lib
$ PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/
    pkgconfig
$ export PKG_CONFIG_PATH
```

B. Possible Problems

During the installation process of OpenCV, several problems might be encountered. In this tutorial, we will discuss the possible errors that might block the installation procedure, and their corresponding solutions.

IV. M3WSN Framework Installation

A. Downloading M3WSN

The M3WSN framework can be downloaded from http://cds.unibe.ch/research/M3WSN/index.html. After downloading, unzip the file, which includes a M3WSN folder and a customized version of Castalia.

B. Building the Project

1. Import Castalia-3.2 M3WSN and M3WSN project to OMNeT++ IDE.

2. Check if M3WSN has reference to Castalia-3.2 M3WSN.

3. Clean and build the project.

V. Creating Video Sequences

Applications involving multimedia transmission must be evaluated by measuring the video quality level from the user's perspective. Due to the importance of the multimedia content, it is essential to visually determine the real impact of the event, perform object/intruder detection, and analyze the scenes based on the collected visual information. Specifically, frames with different priorities (I, P and B) compose a compressed video, and the loss of high priority frames causes severe video distortion from humans experience. For the loss of an I-frame, the errors propagate through the rest of the Group of Picture (GoP), because the decoder uses the I-frame as the reference frame for other frames within a GoP. However, Castalia framework, including both WiSE-MNet and WVSN models, does not enable the transmission, controlling and evaluation of real video sequences. To this end, we ported Evalvid [3] for the M3WSN framework. Evalvid provides video-related information, such as frame types, received/lost frames, delay, jitter, and decoding errors, as well as inter and intra-frame dependency of the received/distorted videos. These information will be helpful to design new video transmission protocols.

Evalvid is a framework for video transmission and quality evaluation. Therefore, before transmitting a real video sequence, we need a video source, for instance from a video library or the user can create a new one. Once the video has been encoded, trace files have to be produced. The trace files contain all the relevant information for video transmission, and the evaluation tools provide routines to read and write these traces files for multimedia evaluation. Information about how to create video traces using Evalvid can be found in http://www2.tkn.tu-berlin.de/research/evalvid/EvalVid/docevalvid.html.

VI. Creating Mobility Traces

Mobility is one of the most challenging issues in WMSNs. To understand how the network behaves under different mobility situations, the node mobility has to be simulated in a reasonable way. In this context, we have ported BonnMotion to M3WSN to support mobility. BonnMotion is a simulator-independent tool to generate mobility traces for different mobility models. It provides several mobility models, such as the Random Waypoint model, the Gauss-Markov model, and others. The generated mobility traces can be exported to compatible simulator. Information about how to create mobility traces can be found at http://sys.cs.uos.de/bonnmotion/.

VII. Running Experiments

Experiments can be run using both simulation IDE or using command line. We show two approaches in below.

A. Using OMNeT++ IDE

After successfully importing and building the project, simulations can be run by following the next steps:
1. Open some configuration files (ini file) from M3WSN/Simulations folder to run the simulations. The configuration file (ini file) describes the experiment scenario and it includes all the settings of the involved modules, i.e., applied protocols at different network layers, protocol parameters, simulation duration, etc. For instance, routing protocol/parameters, application module/parameters, and radio module/parameters can be configured as below:

```
$SN.node[*].Communication.RoutingProtocolName = "
    GPSR"
$SN.node[*].Communication.Routing.netBufferSize =
    64
$SN.node[*].Communication.Routing.
    netDataFrameOverhead = 6

$SN.node[1].Communication.Radio.TxOutputPower = ${
    TxPower="-5dBm", "-10dBm", "-15dBm"}
$SN.node[0].Communication.Radio.CCAthreshold = ${
    CCAthreshold=-95, -90, -85}

$SN.node[*].ApplicationName = "ThroughputTest"
$SN.node[*].Application.packet_rate = 5
$SN.node[*].Application.constantDataPayload = 2000
```

2. Select "Command line" in the "Options" before running the experiments, as shown in Figure 1.

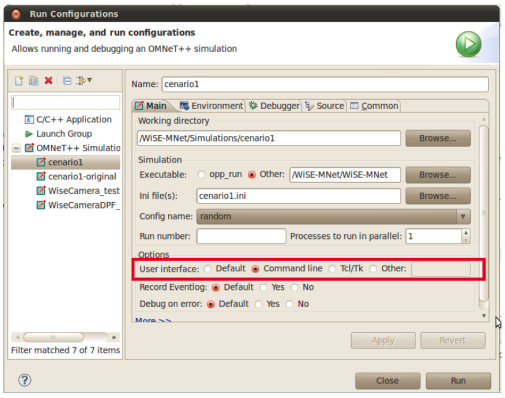


Fig. 1. Simulation configuration using IDE

B. Using command line

In addition to running the experiment using OMNeT++ IDE, it is also possible to run simulation using command line. After building the project using IDE, the scripts at M3WSN/Simulations folder can be used to start the experiments. Additionally, M3WSN framework provides scripts to reconstruct video sequence, measure video quality level, and run simulations with different random-generated seeds.

VIII. Simulation Outputs

After the simulation is finished, results will be saved in different files.

A. M3WSN-result.txt

The file M3WSN-file.txt is automatically generated as done in Castalia. It contains a summary of simulation procedure. The user can use some scripts to filer the results, which can be found in section 3.3 of Castalia user manual.

This file is used to generate customized outputs by using "output()" function to the *.cc* file. Then, the user can use some external tool to analyze the file, and extract some results from them. By default, all tracing is turned off, and the user can turn on for each module in the *.ini* file. For example:

```
$ SN.node[*].ResourceManager.collectTraceInfo =
    True
$ SN.node[*].SensorManager.collectTraceInfo =
    False
$ SN.node[*].Communication.Routing.
    collectTraceInfo = True
$ SN.node[*].Communication.MAC.collectTraceInfo =
    True
$ SN.node[*].Communication.Radio.collectTraceInfo
    = True
$ SN.wirelessChannel.collectTraceInfo = True
```

An example of M3WSN-result.txt is shown as below:

```
Transmitted Videos
Time        Video-id        Node        Hops
4.13594        0             1           5
24.1421        1             1           7
44.1471        2             1           6
64.1513        3             1           6
84.2111        4             1           6
104.154        5             1           6
```

B. M3WSN-Debug.txt

It contains a trace of all events that the user has requested to be recored by "turning on" some parameters in the *.ini* file. This file can be used to debug the code. The user has to use the command "trace()" in the *.cc* file to add information into this file. By default, all tracing is turned off, and the user can turn on for each module in the *.ini* file as for the M3WSN-result.txt

Here we also give show an example of M3WSN-Debug.txt:

```
0.027540267327 SN.node[1].Application Node 1 is
    sending packets
3.868523146136 SN.node[0].Application Received
    packet #18 from Node 1
4.062451653241 SN.node[0].Application Received
    packet #19 from Node 1
4.263512358156 SN.node[0].Application Received
    packet #20 from Node 1
4.463984107516 SN.node[0].Application Received
    packet #21 from Node 1
4.668401238515 SN.node[0].Application Received
    packet #22 from Node 1
```

From the generated output files, we could use some scripts to extract the information of interest, such as to calculate the packet delivery ratio of a node by checking the number of transmitted packets and the number of packets successfully received by other nodes.

IX. Performance Evaluation

In this section, we introduce a use case that makes use of M3WSN framework to obtain key video-related information, such as frame type and GoP length for creating new assessment and optimization solutions. Additionally, the described use case shows the importance to evaluate the transmitted video sequences from the user's perspective. This use case scenario can be easily extended to smart cities applications.

A. Scenario Description

Multimedia video transmission is applicable to many situations, such as multimedia surveillance, real-time traffic monitoring, personal health care, and environmental monitoring. In this tutorial, we apply M3WSN framework in an intrusion detection scenario where static scalar and camera sensors are deployed to monitor a corridor to detect any intruder. The intrusion detection approach is based on our proposed QoE-aware FEC (Forward Error Correction) mechanism for intrusion detection in multi-tier WMSNs, which was described in [4]. In the scenario, a set of sensors performs intrusion detection using vibration sensors. Another set of camera nodes only transmit real-time videos from the intruder area, once the scalar sensors detected it. At the camera nodes, the QoE-aware FEC mechanism creates redundant packets based on frame importance from user's experience, and thus reduce the packet overhead, while keeping the video with a good quality.

On the basis of the multi-tier intrusion detection scenario, simulation can be carried out to evaluate the transmitted video from user's perspective by using QoE-aware FEC. Following this, we simulated a simple FEC approach, i.e., creating redundancy for all the frames (simple FEC), and also without any FEC mechanism (no-FEC). The simulations were carried out and repeated 20 times with different random seed numbers to provide a confidence interval of 95%. Table I shows the simulation parameters for these solutions.

TABLE I
Simulation Parameters

Parameter	Value
Field Size	80x80
Location of Base Station	40, 0
Initial location of intruder	0, 0
Intruder movement type	Random mobility
Intruder velocity	1.5
Total number of Nodes	100
Number of nodes at high-tier	25
High-tier deployment	Grid
Low-tier deployment	Uniform
Transmission Power	-15 dbm
Path loss model	Lognormal shadowing model
Radio model	CC2420
Video sequence	Hall
Video Encoding	H.264
Video Format	QCIF (176 x 144)
Frame Rate	26 fps

The intruder starts at location (0,0), and moves in a random way. The low-tier nodes have an ominidirectional sensing range, and detect the intruder by using the intruder bounding boxes. As soon as the low-tier detects the intruder, it must wake up the high-tier to send the video of the detected intruder. Video flows provide more precise information for users and authorities (e.g. the police) about the intruder, and enable them to monitor, detect, and predict the intruder's moving direction. Additionally, they allow the authorities to take precise actions in accordance with the visual information.

B. Performance Metrics

Existing works on multimedia area classify the videos into three categories, according to their motion and complexity, i.e. low, median and high. For example, Aguiar et al. classify the Hall video sequence (taken from the Video Trace Library) as low movement, which means that there is a small moving region on a static background, i.e. men walking in a hall [**?**].

We evaluated the transmitted videos by means of two well-known objective QoE metrics, i.e. Structural Similarity (SSIM) and Video Quality Metric (VQM), obtained by using the MSU Video Quality Measurement Tool (VQMT) [5]. SSIM measures the structural distortion of the video, and attempts to obtain a better correlation with the user's subjective impression. SSIM has values ranging from 0 to 1, a higher value meaning a better video quality. On the other hand, VQM measures the "perception damage" of video experienced, based on features of the human visual system, including distinct metric factors such as blurring, noise, color distortion and distortion blocks. A VQM value closer to 0 means a video with a better quality.

C. Simulation Results

In the experiments, we measure the SSIM and VQM for transmitted videos with respect to the length of the transmission route (hop numbers), as shown in Figure 2 and Figure 3

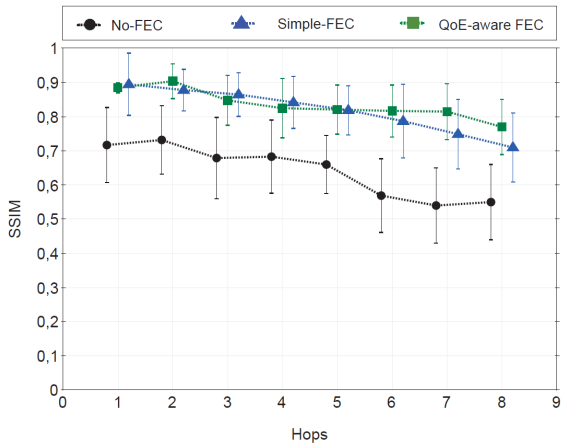


Fig. 2. SSIM with respect to number of hops

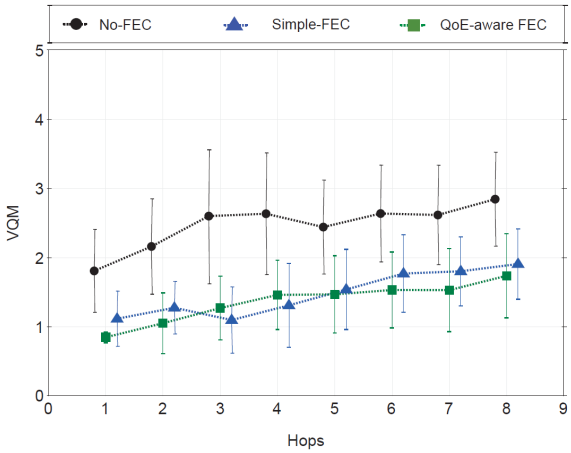


Fig. 3. VQM with respect to number of hops

Figure 2 shows that solutions create redundant packet improve the SSIM by around 25% compared to solutions without FEC. This is due to the fact that application-level FEC is applied as error control scheme for handling packet losses in real-time transmissions. Hence, the redundant packets can be used to reconstruct a lost frame, and thus improve the video quality from a user's perspective. Due to less transmission means less energy consumption, we can conclude that QoE-aware FEC can provide energy-efficiency, while keeping the transmitted video with a good quality. Tis is because QoE-aware FEC creates redundant packets based on frame importance and user experience to reduce network overhead.

Figure 3 presents the video quality by using VQM. The VQM results demonstrate the benefits of using FEC and confirm the SSIM values. Both simple and QoE-aware approaches kept the VQM vales below the solution without FEC, and thus improve the video quality level. However, the QOE-aware FEC mechanism reduces the amount of generated redundant packet while keeping videos with an acceptable quality level.

Last, to show the impact of transmitting video streams from the standpoint of an end-user, a frame was randomly selected (Frame 258) from the transmitted video, as displayed in Figure 4. Frame 258 is the moment when a man (the intruder in our application) was walking along a corridor. For intruder detection application, this is an important frame to provide users and authorities with more precise information and allow them to make actions. The benefits of the FEC mechanisms are visible by analyzing the frames in Figure 4. By comparing each transmitted frame with the original one, it is possible to see a higher distortion for the frame transmitted without using any FEC, as shown in 4(a). The frames transmitted using FEC mechanism achieves low distortion, as shown in 4(c) and 4(d). The visual evaluation is only possible due to M3WSN supports the transmission and control of real video sequences.

(a) Original Frame (b) No FEC

(c) QoE-aware FEC (d) Simple FEC

Fig. 4. Frame 258 of transmitted video

X. Conclusion

This tutorial gives detailed explanations about how to install and configure the M3WSN OMNeT++ framework, which enables the transmission of real video sequence. M3WSN framework supports mobility, and it can generate real video sequences. During the tutorial, we also discuss the possible problems that might be encountered during the installation process and the corresponding solutions.

M3WSN can also be used to evaluate protocols at different network stacks, e.g., routing protocols, transport protocols, or audio/video codes mechanisms.

References

[1] D. Rosario, Z. Zhao, C. Silva, E. Cerqueira, and T. Braun, "An omnet++ framework to evaluate video transmission in mobile wireless multimedia sensor networks," in *Proceedings of the 6th International Workshop on OMNeT++*, Cannes, France, March 2013.

[2] C. Nastasi and A. Cavallaro, "Wise-mnet: an experimental environment for wireless multimedia sensor networks," *Proceedings of Sensor Signal Processing for Defence (SSPD)*, 2011.

[3] OpenCV, "Open source computer vision library," available at: http://opencv.org/. Accessed at June 2015.

[4] Z. Zhao, T. Braun, D. Rosario, E. Cerqueira, R. Immich, and M. Curado, "Qoe-aware fec mechanism for intrusion detection in multi-tier wireless multimedia sensor networks," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*, Oct. 2012.

[5] "Msu quality measurement tool," available at: http://goo.gl/Ng4eGI.

Panel Discussion: Simulating power consumption in OMNeT++

Laura Marie Feeney, Panel Organizer
Swedish Institute of Computer Science and Uppsala University
http://www.sics.se/~lmfeeney

***Abstract*—A panel discussion on simulating power consumption was organized for the OMNeT++ Summit. Six speakers are scheduled to discuss their experiences and thoughts for the future.**

Energy efficiency has become an important performance metric for an increasingly wide range of systems – from networks of small battery-powered radios that consume only a few mW of power to large data centers that can consume a MW or more. Unsurprisingly, models of energy sources and power consumption have been developed for a wide range of OMNeT++-based simulators.

This panel discussion is intended to highlight the state-of-the-art, as well as some of the main issues around simulating power consumption in OMNeT++. One goal of this session is to stimulate further discussion within the community about how OMNeT++ can better support this important functionality and to encourage collaboration among researchers and developers.

Six speakers are currently scheduled to participate in the panel. Several of them have also provided Invited Abstracts, which are included in the Summit proceedings:

Saeed Bastani, Lund University
Content Delivery Networks (CDNs) are becoming an integral part of the future generation Internet. Traditionally, these networks have been designed with the goals of traffic offload and the improvement of users' quality of experience (QoE), but the energy consumption is also becoming an indispensable design factor for CDNs to be a sustainable solution. To study and improve the CDN architectures using this new design metric, we are planning to develop a generic and flexible simulation package in OMNet++. This package is aimed to render a holistic view about the CDN energy consumption behaviour by incorporating the state-of-the-art energy consumption models proposed for the individual elements of CDNs (e.g. servers, routers, wired and wireless links, wireless devices, etc.) and for the various Internet contents (web pages, files, streaming video, etc.).
Saeed Bastani is a post-doctoral researcher at Lund University.

Torsten Braun, University of Bern
Our work addresses issues with state-based energy-consumption modeling: Energy consumption is usually modeled by a state based approach with constant energy consumption values in each state. However, it happens in certain situations that during state changes or even during a state the energy consumption is not constant. We will present some examples for such cases and possible solutions.
Torsten Braun is a professor at the University of Bern.

Radu Carpa, ENS Lyon
I seek to reduce the energy consumption of the networks while maintaining a high level of quality of service. I focus on backbone and inter-datacenter networks and use intelligent and increase its energy efficiency.
Radu Carpa is a PhD student under the supervision of Laurent Lefèvre and Olivier Glück at LIP laboratory - Avalon Inria team - ENS Lyon.

Laura Marie Feeney, SICS/Uppsala University
The EnergyFramework provides abstractions for simulating energy consuming device components and an energy store, as well as statistics collection. The goal of this structure was to make it possible to incorporate power consumption modeling into a wide range of simulation models. It was partly successful: EnergyFramework has been ported to several wireless network frameworks, most notably MiXiM and its descendants. We consider lessons learned and possible directions for development of a more general, flexible and easily integrated structure.
Laura Marie Feeney is a researcher at the Swedish Institute of Computer Science and Uppsala University.

Sei Ping Lau, University of Southampton Sei Ping Lau is working on enabling energy savings in networked streetlights with StreetlightSim, a simulation environment based on OMNeT++.
Sei Ping Lau is at the University of Southampton.

Dora Spenza, University of Rome – La Sapienza
GreenCastalia is an open-source extension to the Castalia simulator for energy-harvesting Wireless Sensor Networks. GreenCastalia supports multi-source and multi-storage energy-harvesting architectures and it allows to simulate networks of embedded devices with heterogeneous harvesting capabilities.
Dora Spenza is a post-doctoral researcher at the University of Rome, La Sapienza.

Invited Abstract: Issues with State-based Energy Consumption Modelling

Torsten Braun, Philipp Hurni
University of Bern
Communication and Distributed Systems
Bern, Switzerland
{braun, hurni}@inf.unibe.ch

Vitor Bernardo, Marilia Curado
University of Coimbra
Center for Informatics and Systems
Coimbra, Portugal
{vmbern, marilia}@dei.uc.pt

***Abstract* — Energy consumption modelling by state based approaches often assume constant energy consumption values in each state. However, it happens in certain situations that during state transitions or even during a state the energy consumption is not constant and does fluctuate. This paper discusses those issues by presenting some examples from wireless sensor and wireless local area networks for such cases and possible solutions.**

Keywords — Simulation, Wireless Networks, Energy Consumption

I. Introduction

Energy consumption is an important issue in wireless networking, in particular in sensor networks with sensor nodes operating for long periods with a single battery as well as in mobile end systems such as smart phones, where users like to enjoy many hours of battery driven operation. Protocols do not only have to support traditional performance metrics such as delay and throughput, but also should support energy-efficient operation. Protocols have to be evaluated in real-world testbeds and simulators. Simulators aim to support accurate energy consumption evaluation by relying on accurate energy models for the wireless device's transceiver, which is often by far the most energy-consuming component in a wireless device.

Typically, energy consumption in simulators is modelled by rather simple state-based approaches, where the time T_i of the transceiver being in a state i is recorded and multiplied with the average current I_i in state i as well as the (often constant) supply voltage U to calculate the consumed energy. We further have to sum up the energy consumed in each state I_i as depicted in Fig. 1. The following formula calculates the overall energy E consumed by the system:

$$E = \sum_{i=1}^{N} T_i \cdot I_i \cdot U$$

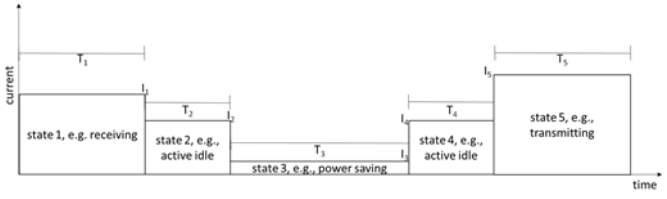


Fig. 1. State-based energy consumption modelling

However, this state-based energy consumption model is only an approximation of the energy consumption in a real system and has some limits in terms of accuracy. As discussed in subsequent sections of this paper the state-based model is somewhat inaccurate, because it is not correct to assume that the current is constant during all states, state transitions do take some time, and the current during the state transition is not equivalent to the current in the previous or subsequent state. Those effects can then lead to some inaccuracy in energy consumption evaluation, as we will discuss using two example scenarios in Sections II and III.

II. Energy Consumption in Wireless Sensor Networks

In this section, we discuss some energy consumption modelling for wireless sensor nodes [1]. In Fig. 2 we show the measurement of the current flow of a sensor node (MSB430), which is forwarding traffic from a source to a destination node. We identify three states of the node's transceiver, namely sleep, transmitting, and receiving. Fig. 3 shows how the energy consumption of a sensor node is modelled using a so-called three-state-model. Here, the model is not used for simulation but for so-called software-based energy estimation, where the running software of a node measures the time of the transceiver in a state and multiplies the time with the corresponding average current in a state and the supply voltage, similar as discussed in Section I.

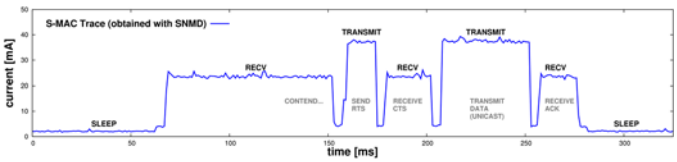


Fig. 2. Current draw of a sensor node [1]

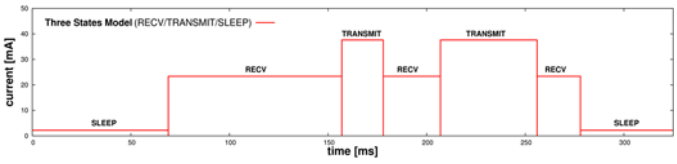


Fig. 3. Sensor node modelled by the three-states model [1]

The software-based energy estimation has been used for the evaluation of several MAC protocols (CSMA, TMAC; SMAC; and WiseMAC) in a wireless sensor network

consisting of three nodes, i.e. a source, a forwarder and a destination node, with variable traffic rate. In Fig. 4, dashed and solid lines represent estimated and measured energy consumption, respectively. We can observe that for increasing the traffic rate, measured in packets/s the difference between estimation based on the three-state model and the real evaluation becomes larger. The reason is that for increasing traffic rate the number of state transitions increases too. Comparing Fig. 2 and Fig. 3, we see that in reality there is lower energy consumption during a state transition than what we assume in the software-based estimation using the three-state-model.

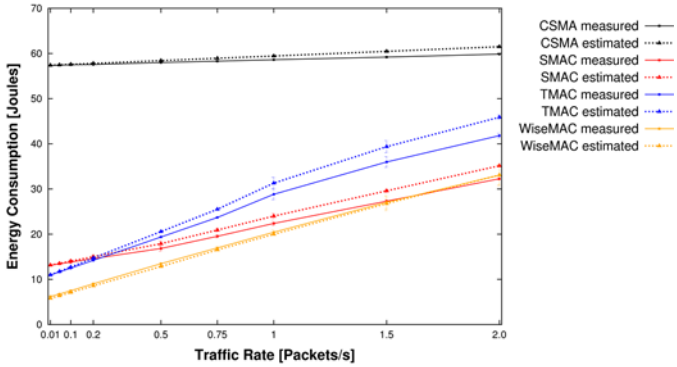


Fig. 4. Measured vs. Estimated Energy Consumption [1]

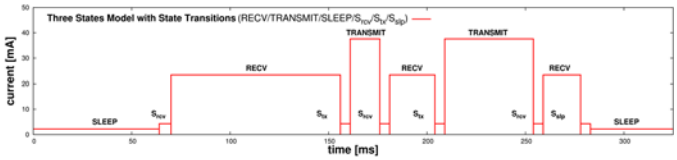


Fig. 5. Current modelled by the three-states model with state transitions [1]

In order to improve energy consumption estimation, we introduce a better model considering the energy consumption during state transitions, cf. Fig. 5. Fig. 6 shows the measured error of the original three-states model, an improved version of the three-states model using ordinary least square (OLS) regression analysis for estimating energy consumption values of the various states, and the OLS-based three-states model considering state transitions. Fig. 6 shows that by considering state transitions we can improve the estimation accuracy from approximately 4 % to 1 % for the OLS-based three-state model without and with considering state transitions. While the improved model has been applied for software-based energy estimation [1], we could also apply state transitions in a simulation model for more accurate energy consumption modelling.

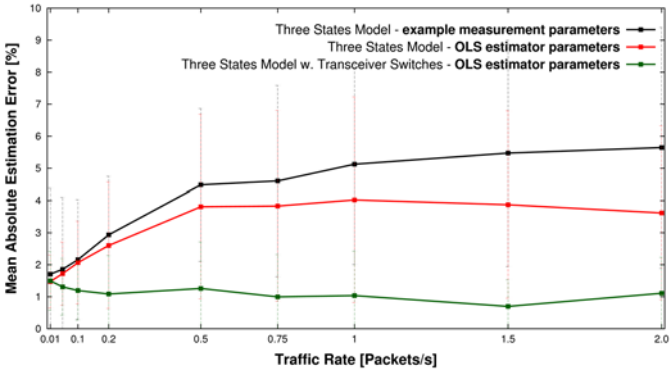


Fig. 6. Absolute Mean Estimation Error (in %) vs. Traffic Rate (packets/s)[1]

III. Energy Consumption in Wireless Local Area Networks

The IEEE 802.11 standard [3] defines a power management mode that allows the mobile stations turning off both transmitter and receiver capabilities in order to save energy. Fig. 7 shows the simplified state diagram of IEEE 802.11 nodes with power management. Each state consumes a different amount of energy.

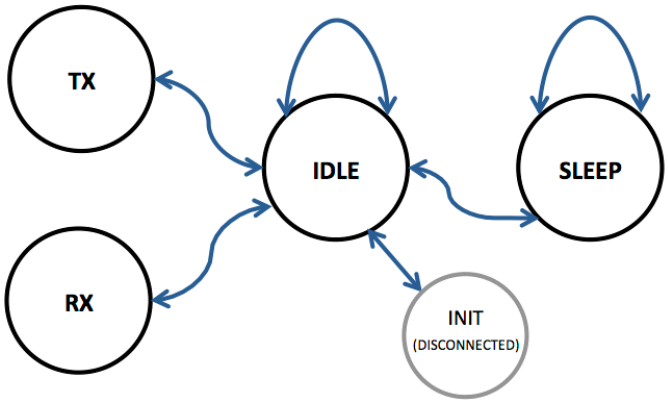


Fig. 7. Simplified IEEE 802.11 state diagram [2]

Fig. 8 shows the average power consumed (in mW) by an IEEE 802.11 network card (Linksys TP-LINK WN-721n operating in the 2.4GHz frequency band) in three distinct states, namely disconnected, idle and sleep. The depicted values were calculated as average of the 20 runs performance evaluation for each test setup with confidence intervals of 95%. By analysing the error bars for each state, one can observe a lower uncertainty for all the states, showing the suitable accuracy achieved with the employed measurement methodology. See [2] for details about the experiments.

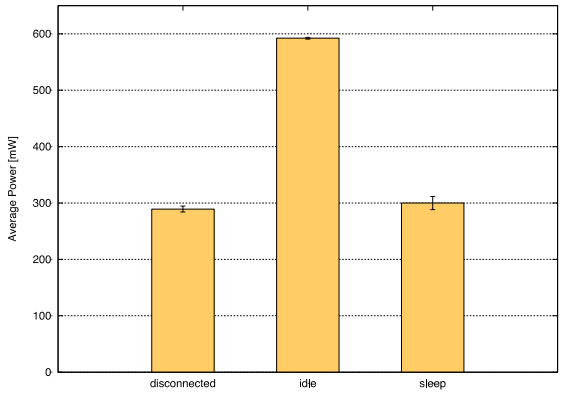


Fig. 8. Average power in disconnected, idle, and sleep states (adapted from [2])

Unlike when operating in disconnected, idle and sleep states, power consumption in the receiving and transmitting states is less constant, as it is depicted in Fig. 9 and Fig. 10. Fig. 9 shows the measured power consumption of the same IEEE 802.11 network card without enabled power management. Fig. 10 shows the power consumption of the same network card with enabled power management. Fig. 9 and Fig. 10 show strong fluctuations during sending/receiving dependent on the network traffic. Moreover, there is some phase (connecting), where the node is moving from disconnected towards connected state. We can compare this phase to a state transition as discussed in Section II. Again, we see that the energy consumption during this phase/state transition is somewhat different from the previous and subsequent state. In this case, energy consumption during state transition is somewhat lower (due to negative peaks) than in the connected state, but significantly higher when compared to the disconnected state.

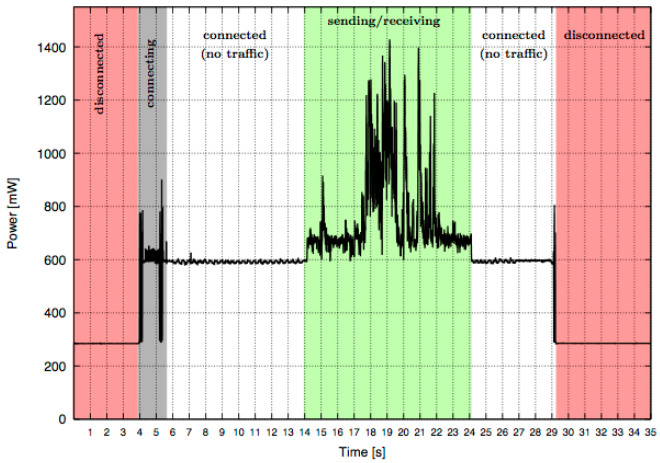


Fig. 9. IEEE 802.11 network card power consumption without power management [2]

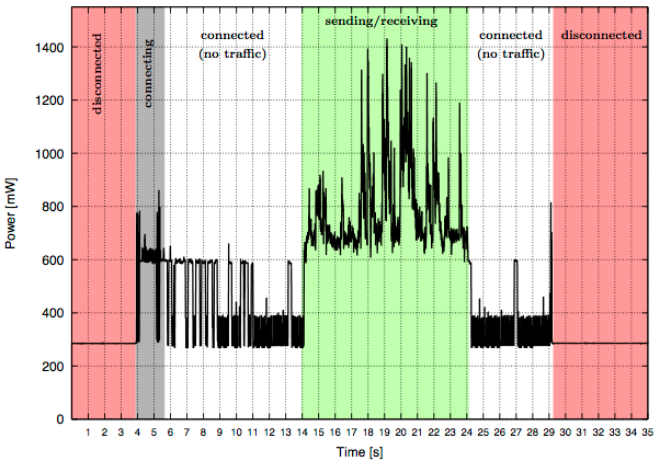
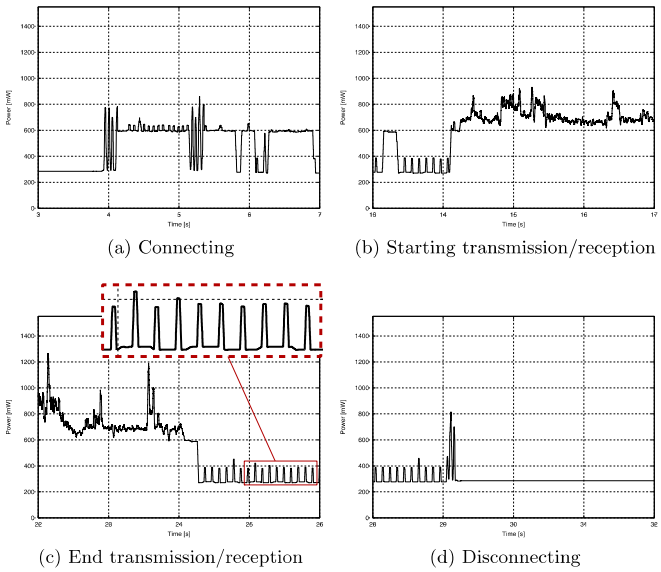


Fig. 10. IEEE 802.11 network card power consumption with power management [2]

Fig. 11 zooms four state transitions of the power consumption measurements from Fig. 10, namely connecting (Fig. 11a), starting transmission/reception after being connected without traffic (Fig. 11b), stopping transmission/ reception before being connected without traffic (Fig. 11c), and disconnecting (Fig. 11d). In Fig. 11 b-d, we see some periodic peaks while being connected without traffic. This is further zoomed in Fig. 11c. Those peaks are caused by receiving beacon frames from the access point. In this scenario beacons are configured to be sent by the access point every 100 ms. The peaks increase the average energy consumed and such effects have to be considered for accurate energy modelling.



(a) Connecting (b) Starting transmission/reception

(c) End transmission/reception (d) Disconnecting

Fig. 11. Detailed states transition power consumption pattern with power saving enabled [2]

IV. Discussions and Conclusions

In this paper, we have presented energy measurement results from previous work on energy consumption evaluations in wireless sensor and wireless local area networks. We have made the following observations:

1. Energy consumption during state transitions can significantly differ from previous and subsequent states.
2. During an active state, e.g., transmitting, receiving, active idle/connected without traffic, the energy consumption can vary dependent on the current traffic transmitted or received. This also includes reception of control messages, as it is the case for IEEE 802.11 beacons.

For accurate evaluation of energy consumption in either software-based energy estimation or simulation, where state-based energy consumption models have been applied in the past, we might need to more accurately model state transitions and dynamic fluctuations within a state. In particular, in [1] we have shown that we can significantly decrease the estimation error by considering the behaviour during a state transition. Further improvements might result from considering other parameters such as the number and size of received/transmitted data/control messages during a state. More measurements of specific wireless network hardware are needed to investigate the impact of such parameters to energy consumption.

References

[1] Philipp Hurni, Benjamin Nyffenegger, Torsten Braun, Anton Hergenroeder: On the Accuracy of Software-Based Energy Estimation Techniques, Wireless Sensor Networks, Lecture Notes in Computer Science Volume 6567, Springer-Verlag, 2011, pp. 49-64

[2] Vitor Bernardo, Marilia Curado, Torsten Braun: An Overview of Energy Consumption in IEEE 802.11 Access Networks, in Wireless Networking for Moving Objects, A. Kassler, I. Ganchev, M. Curado (eds.), Lecture Notes in Computer Science Volume 8611, Springer-Verlag, 2014, pp. 157-176

[3] IEEE 802.11 Standards Association: IEEE Standard for Information technology - Telecommunications and information exchange between systems, Local and metropolitan area networks - Specific requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std. 802.11-2012 (Revision of IEEE Std 802.11-2007)

Invited Abstract: A simulation package for energy consumption of content delivery networks (CDNs)

Mohammadhassan Safavi Saeed Bastani
Department of Electrical and Information Technology, Lund University, Lund, Sweden
{*mohammadhassan.safavi, saeed.bastani*}*@eit.lth.se*

***Abstract*—Content Delivery Networks (CDNs) are becoming an integral part of the future generation Internet. Traditionally, these networks have been designed with the goals of traffic offload and the improvement of users' quality of experience (QoE), but the energy consumption is also becoming an indispensable design factor for CDNs to be a sustainable solution. To study and improve the CDN architectures using this new design metric, we are planning to develop a generic and flexible simulation package in OMNet++. This package is aimed to render a holistic view about the CDN energy consumption behaviour by incorporating the state-of-the-art energy consumption models proposed for the individual elements of CDNs (e.g. servers, routers, wired and wireless links, wireless devices, etc.) and for the various Internet contents (web pages, files, streaming video, etc.).**

I. Introduction

With the tremendous growth of the Internet traffic and the demand for guaranteed quality of service, the content delivery networks (CDNs) have emerged as a promising solution to offload the traffic and to reduce content access delay, so that both content providers and consumers benefit directly. There is also a new emerging design goal for CDNs, motivated by the present trend of energy consumption in the ICT sector which implies for an urgent need for re-assessing and re-designing Internet systems to achieve energy efficient architectures. In particular, the content delivery networks can play a significant role in harnessing energy consumption per bit delivered to the end users. This is supported by the fact that the Internet is experiencing an unprecedented growth in the type and volume of contents. The most significant phenomenon of this type is video traffic. A recent trend of the Internet usage indicates an increased demand for streaming media (IPTV, Youtube etc.) in the downlink as well as an increased user generated traffic (upload of video to Facebook, Youtube, Vimeo etc.) in the uplink side. Knowing the fact that the delivery of video contents is highly bandwidth and processor demanding, it is of prominent value to understand how the current CDNs behave in terms of energy efficiency, and how to calibrate the existing CDNs or to design new ones by taking into account the energy efficiency as an additional design metric. The study of energy efficiency of the existing CDNs and, more importantly, the design of new CDNs require that the contributions of a tremendous number of factors to the energy consumption is taken into consideration. On one side, the end-users are increasingly inclined to use battery powered mobile devices or portable computers. On the other side, there are a huge number of network elements (e.g. servers, storages, routers, links, etc.) responsible for providing, managing, and carrying the Internet contents to the users. Therefore, the performance study of a given CDN architecture requires that the entire set of network elements with a role in carrying contents from a content provider to the consumers are taken into account. Moreover, contents have different types and properties and users have different preferences over contents and use different devices to access those contents. With this extremely huge set of elements involved in a CDN, simulation as a tool appears as the most affordable way of studying the energy efficiency of the existing CDNs, and to design new CDNs with respect to energy efficiency metric.

In the following, we present the building blocks of an envisioned simulation package for the studying of energy consumption behaviour of CDNs.

II. A Simulation Package for Energy Consumption Behaviour of CDNs

Our proposed simulation package is depicted in Figure 1. The description of the building blocks of the proposed package is as follows:

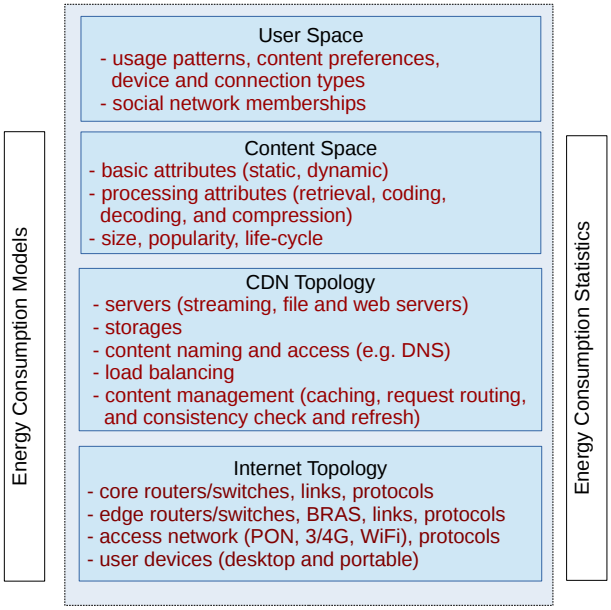


Fig. 1: Building blocks of the proposed simulation package

- **User Space:** this module defines the various attributes of a user including the frequency and time patterns of user access to contents of various types. Also, it defines the type and properties of the device(s) used by the user and the way they connect to the network. The device and connection type determine the amount of energy consumed in the user premise to access a unit of content. Additionally, the user's social activities are used to predict the frequency by which the user will access the contents shared by the members of her social network. This latter information is used by the content management component of the CDN to efficiently cache and replicate those contents deemed to become popular.
- **Content Space:** it defines the contents attributes including the set of content types and their (relative) popularities extracted from field data or described by appropriate distribution functions. It also defines the life-cycle of each content characterized by the time period the content is present in the network and its popularity pattern during its lifetime. Another attribute of a content which directly relates to energy consumption is the resources (bandwidth and processor time) required to carry the content from its source to the user device. Apparently, this property depends on factors other than the content itself, including the types and properties of the network elements involved in processing and carrying the content. However, in this module the focus is on the abstract features of the content, e.g. required bitrate and the coding and decoding complexity of the (video) content regardless of the hosting devices.
- **CDN Topology:** this module defines the network topology and the enabling functions/tasks of the CDN. Two different groups of attributes are specified in this module: the hardware and software. The hardware attributes defines the the type, capacity, and the power density of the servers and the storage devices. Other hardware elements are defined in this group including the networking equipment (switches) and links to interconnect the servers and storages internally and externally with the rest of the network. The software attributes are specified by the content management tasks (as a major task of the CDN), content addressing and request routing, and load balancing function.
- **Internet Topology:** it defines the network equipment and the interconnection devices and protocols, altogether representing the Internet as the underlying network for CDNs.
- **Energy Consumption Statistics:** this module implements statistic collection functions and the GUIs needed to demonstrate and analyse the energy consumption behaviour by the individual network layers, and the energy efficiency of the CDN in its entirety.
- **Energy Consumption Models:** this module is regarded as the core part of the envisioned simulation package. It relies on the energy consumption models proposed in the literature. These models cover a wide range of CDN elements from processing units, cooling equipment, storage, and communication interfaces. This module takes as input those features of a typical content which describes, for example, the time complexity of the content processing (e.g. streaming, coding, decoding), and combines them with the power density of a given network element (e.g. a processing unit of a server) currently hosting the content, and calculates the amount of energy consumed to perform the required operation on the content.

In the following, we briefly describe some specific considerations about the CDN architecture subject to implement in the simulation package and the energy consumption models to be incorporated in this package.

In the traditional CDN architecture, there is a single set of surrogate (or delivery) servers located in a single architectural level, but in the recent CDNs the surrogate servers are located in multiple levels to render a hierarchical architecture [1] (see Figure 2). Due to the generalization concerns, we will concentrate on this reference architecture in our simulation package.

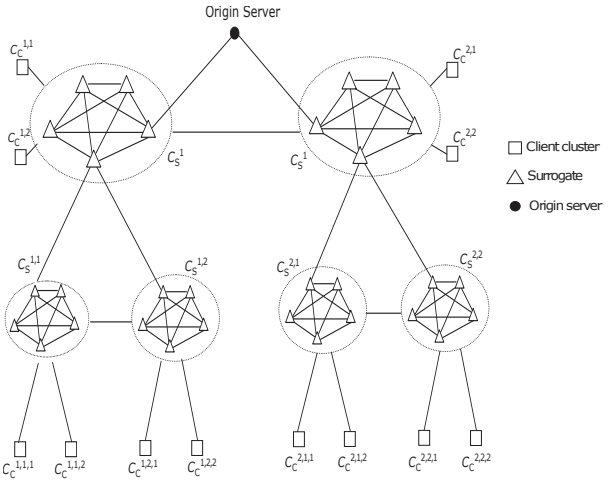


Fig. 2: A state-of-the-art CDN model

In the reference CDN (Figure 2), C_S is server cluster and C_C is client cluster, and the superscripts refer to the index of the individual clusters and systems.

It is ideal to have end-to-end energy consumption models in place and incorporate them in the simulation package. However, such models have not been developed for the reference CDN architecture depicted in Figure 2, and the existing models each describes a subset of the elements in the end-to-end path from the source to the user. Therefore, we opt to select some appropriate models of this kind and integrate them into a holistic end-to-end energy consumption model. The most complete model so far has been proposed in [2], which describes the energy consumption of an IPTV network. In this macroscopic model, the energy consumption in watts-hour for

downloading video contents is expressed as follows:

$$E_{download} = 4\frac{B}{3600}\left(\frac{3P_{ES}}{C_{ES}} + \frac{P_G}{C_G} + \frac{2P_{PE}}{C_{PE}} + \frac{(H+1)P_C}{C_C} + \frac{H\,P_{WDM}}{C_{WDM}} + \frac{P_{SR}}{C_{SR}}\right) + 2\frac{BR}{D}\left(\frac{P_{SD}}{S_{SD}}\right) \quad (1)$$

where P is the power density and C is the capacity of the given device. The subscripts ES, G, PE, C, WDM, and SR represent Ethernet switches, gateway routers, provider edge routers, core routers, WDM equipment, and content servers, respectively. The parameters H and B respectively indicate the video content size and the average number of hops between the content server and the end-user(s). R and D represent the number of replicated copies of a content in the data centres and the average number of downloads per hour. From the model, it is evident that with the number of hops H increasing, the energy consumption also increases. For this reason, surrogate servers are deployed with a suitable density to cache the contents and to reduce the distance between the content server and the end-user(s) [3]. The model described above suffers from some limitations, including the lack of differentiation between different access technologies and the absence of specialized models to describe the energy consumption of different content types. To overcome the former limitation, we propose to augment the model with dedicated energy consumption models proposed for wireless access as the most prevalent access technology and the most energy consuming part of the network [4]. In view of this, our first attempt is to incorporate an energy consumption model proposed by [5] into the simulation package. This model describes the power consumption of an 802.11 device by a concise expression as follows:

$$P = \rho_{id} + \rho_{tx}\tau_{tx} + \rho_{rx}\tau_{rx} + \gamma_{xg}\lambda_g + \gamma_{xr}\lambda_r \quad (2)$$

where ρ_{id} is the power consumption in idle mode, ρ_{tx} is transmission power coefficient, ρ_{rx} is reception power coefficient, τ_{tx} is packet transmission airtime, τ_{rx} is packet reception airtime, λ_g is packet generation rate, λ_r is packet reception rate, γ_{xg} is the energy toll associated to the processing of each individual frame at the transmitting device, and γ_{xr} is the energy toll associated to the processing of each individual frame at the receiving device. The model described above is simple and straightforward to implement in OMNet++, e.g. by integrating it into the Energy Framework and the wireless access modules of the INET simulation framework.

The models described by expressions 1 and 2 are mainly focused on the transmission energy, and simplify the processing energy to a great extent. In these models, the packets are regarded as raw data units without taking into account the specific processing features of the given content type. This approach, however, leads to a significant inaccuracy of energy consumption modelling of processor demanding contents such as video streaming. Therefore, our major task in developing the envisioned simulation package is to fill the mentioned gap by using content-specific energy consumption models. Our initial focus in this respect is video contents. Along this line, video coding, decoding and streaming account for the major processing factors of video contents, though these factors are not symmetric in terms of processing complexity and energy consumption behaviour. While video coding is more processor demanding than the decoding, the latter is invoked more frequently (per user request), and thus, from a CDN perspective, it is the dominant factor. An example model of decoding energy consumption is proposed in [6], where the authors derive an energy consumption model from the time complexity of a given codec standard (here H.264 /MPEG 4 AVC). It is generally possible to map from time complexity to energy consumption, on condition that the type of the coding/decoding algorithm (i.e. serial, concurrent, multi-threading) and the speed and architecture of the host processor (single and multi-core) are known. There are also other energy consumption parameters beyond coding/decoding, such as size, resolution and luminance settings of the user's device LCD. The intensity of memory access (i.e. space complexity) is another important factor required to be captured in the desired model.

In conclusion, we believe that OMNet++ yields a high level of flexibility in implementing the proposed simulation package for energy consumption of CDNs. The usefulness of such a package for the networking researchers is substantially dependent on the availability of accurate energy consumption models proposed in the literature. As mentioned previously, there are some promising models such as those described by expressions 1 and 2, which can be incorporated in the first roll-out of the envisioned simulation package. There are also microscopic models, such as [6], which are capable of capturing the essential features of video processing and mapping it to the outcome energy consumption behaviour.

Acknowledgements

This work is funded by the European Celtic-Plus project CONVINcE.

References

[1] Benjamin Molina, Jaime Calvo, Carlos E Palau, and Manuel Esteve. Analyzing Content Delivery Networks. *Advanced Content Delivery, Streaming, and Cloud Services*, pages 203–218, 2014.

[2] Jayant Baliga, Robert Ayre, Kerry Hinton, and Rodney S Tucker. Architectures for energy-efficient IPTV networks. In *Optical Fiber Communication Conference*, page OThQ5. Optical Society of America, 2009.

[3] Vytautas Valancius, Nikolaos Laoutaris, Laurent Massoulié, Christophe Diot, and Pablo Rodriguez. Greening the internet with nano data centers. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 37–48. ACM, 2009.

[4] Uichin Lee, Ivica Rimac, and Volker Hilt. Greening the internet with content-centric networking. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, pages 179–182. ACM, 2010.

[5] Andres Garcia-Saavedra, Pablo Serrano, Albert Banchs, and Giuseppe Bianchi. Energy consumption anatomy of 802.11 devices and its implication on modeling and design. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 169–180. ACM, 2012.

[6] Zhan Ma, Hao Hu, and Yao Wang. On complexity modeling of H. 264/AVC video decoding and its application for energy efficient decoding. *IEEE Transactions on Multimedia*, 13(6):1240–1255, 2011.