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

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

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

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

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

I. Introduction & Problem Statement

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

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

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

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

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

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

II. Background & Related Work

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

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

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

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

III. Simulation Models

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

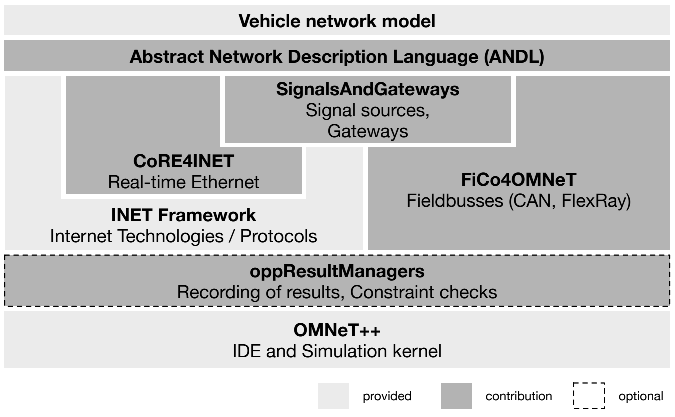
Vehicle network model
Abstract Network Description Language (ANDL)
SignalsAndGateways
Signal sources, Gateways
CoRE4INET
Real-time Ethernet
FiCo4OMNeT
Fieldbusses (CAN, FlexRay)
INET Framework
Internet Technologies / Protocols
oppResultManagers
Recording of results, Constraint checks
OMNeT++
IDE and Simulation kernel
provided
contribution
optional

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

A. CoRE4INET

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

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

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

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

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

B. FiCo4OMNeT

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

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

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

C. SignalsAndGateways

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

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

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

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

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

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

IV. Network Design

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

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

Listing 1
ANDL code example with comments

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

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

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

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

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

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

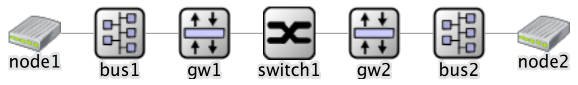

Fig. 2. ANDL generated network consisting of two CAN busses and a real-time Ethernet backbone with two gateways and one switch

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

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

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

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

V. Result Analysis

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

A. Gantt Chart Timing Analyzer

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

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

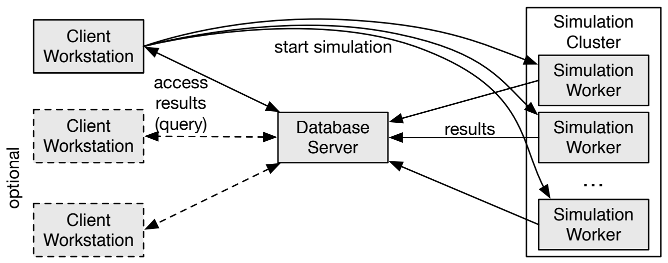

Fig. 3. Storing and accessing simulation results using a database (e.g. postgreSQL).

B. oppResultManagers

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

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

```
eventlogmanager−class = ”PCAPNGEventlogManager”
```

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

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

SQLite is a file based database that is fast and efficient. The SQLite ResultManager is in most cases slightly slower than OMNeT++ vector and scalar managers, but produces smaller result files. The SQLite database is always stored on the machine executing the simulation. As the database is locked when it is opened for writing (similar to a OMNeT++ result file) several simulation processes running concurrently cannot write to the same database. Thus different parameter sets or seeds simulated in parallel will have to use separate SQLite databases. A script provided with the result manager offers to merge several databases to enable queries containing results of several runs. As SQLite databases are regular files on the filesystem they can be easily transferred, archived, or read and manipulated using third party software.

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

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

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

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

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

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

Listing 2
EXAMPLE OF XML DEFINITION FOR THE CONSTRAINT CHECK RESULTMANAGER

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

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

VI. Case Study

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

A. Simulation Workflow

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

B. Simulating the Ethernet Backbone of a Prototype Car

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

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

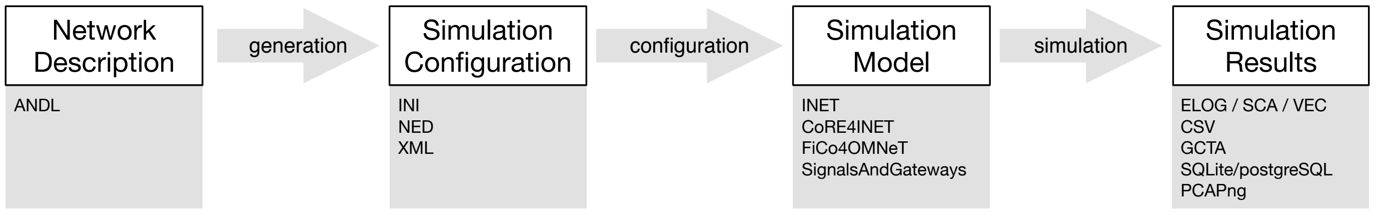

Fig. 4. Workflow of simulation projects – From network description to result analysis

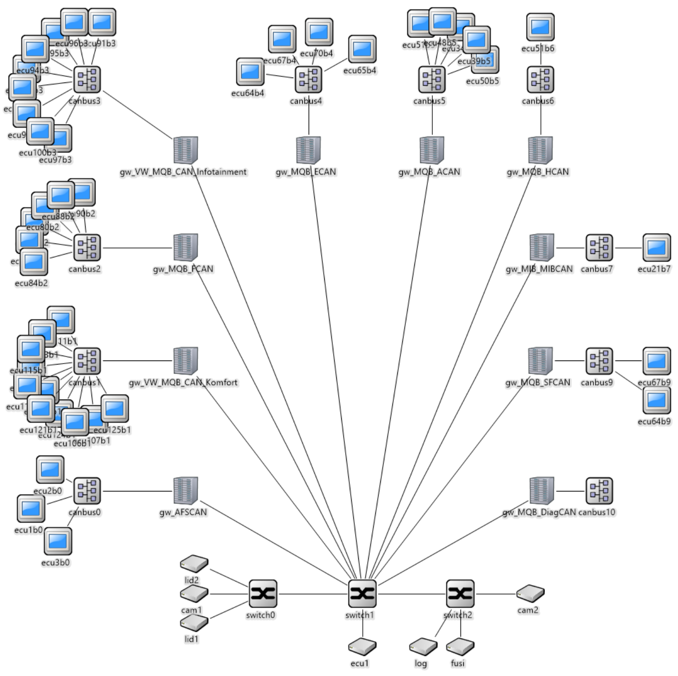

Fig. 5. Simulation of the Architecture of the RECBAR prototype

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

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

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

VII. Conclusion & Outlook

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

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

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

Download

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

http://sim.core-rg.de

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

Acknowledgements

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

References

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

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

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

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

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

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

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

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

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

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