Emulating Vehicular Ad hoc Networks for Evaluation and Testing of Automotive Embedded Systems

Manuel Schiller, Alois Knoll
Robotics and Embedded Systems
Department of Informatics
Technische Universität München
{manuel.schiller,knoll}@in.tum.de

ABSTRACT

The evaluation and testing of cooperative applications based on Vehicular Ad hoc Networks (VANETs) in real testbeds is difficult due to the need for repeatable scenarios and large-scale experiments. Therefore a novel virtualization-based framework is presented to evaluate automotive software in the context of emulated VANETs. The approach enables the precise and large-scale evaluation of real-world implementations through the synchronized execution of network and vehicle simulators as well as the applications encapsulated in virtual Electronic Control Units. This paper provides a detailed description of the framework's structure and its components as well as an validation of the proposed synchronization algorithm. The performance comparison with pure network simulation indicates that despite additional overhead large-scale experiments can be conducted without loss of accuracy.

Keywords

VANET, embedded system simulation, testing, evaluation, network emulation

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

1. INTRODUCTION

Vehicular Ad hoc Networks (VANETs) have attracted a lot of research attention over recent years due to the potential improvements in traffic safety and efficiency as well as driver comfort. A high variety of applications, commonly referred to as Advanced Driver Assistance Systems (ADAS), such as cooperative driving and subsequently automated driving, are enabled through wireless ad hoc communication between the vehicles on the road.

Simulation is currently the key methodology to gain an understanding of the various effects that influence the performance and behavior of the entire system composing a VANET. The majority of publications focuses on exploring the lower-level effects such as wireless signal propagation at the physical layer, medium access control and ad hoc routing protocols. The actual applications which are intended to run on top of these layers are usually either left out completely or are only modeled on a very abstract level. However, these applications, which often exhibit safety-critical features, need to be evaluated and tested extensively before deployment in series production.

While it is theoretically possible to develop simulation models of the actual implementations and execute them in the established simulators, this approach quickly gets infeasible for complex real world applications. At the other end of the spectrum of available methods real world test drives using physical testbeds of prototype vehicles offer the highest degree of realism. Due to the large amount of resources needed for real world test drives, this method is not feasible to perform large-scale and extensive testing of vehicular networks. Additionally, achieving repeatable test conditions is next to impossible. In the automotive industry the use of simulation is well established in the development process of traditional driver assistance and active safety systems. However, the current emphasis is primarily on the simulation of individual vehicles at a very high level of detail [6]. When investigating and evaluating the performance of ADAS based on vehicular communication, this isolated view of a single vehicle in the simulation is not sufficient anymore. Potentially every vehicle equipped with wireless communication technology is coupled in a feedback loop with the other road users participating in the vehicular network. Therefore the number of relevant intelligent entities which need to be taken into account is drastically increased.

To help bridging this gap we present in this paper a new virtualization-based approach for emulating vehicular ad hoc networks as the enabling methodology for evaluating and testing network-centric automotive embedded systems based on this wireless communication technology. Our approach ensures that the actual implementations rather than models are employed in the test procedure taking into account the overall system context. By eliminating the need to create such simplified abstractions, testing can be performed earlier and without potential mismatches between the application and its model.

SIMUTOOLS 2015, August 24-26, Athens, Greece
Copyright © 2015 ICST
DOI 10.4108/eai.24-8-2015.2261004

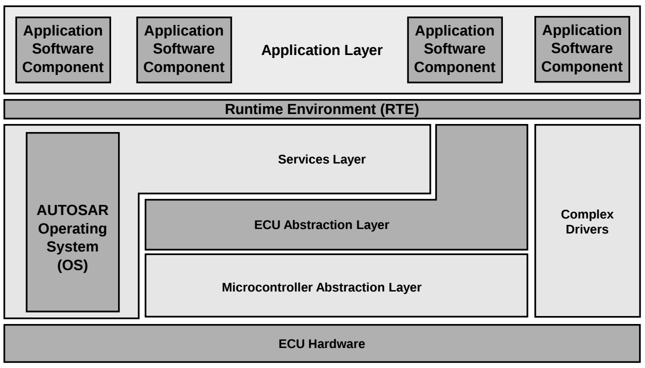
Application Software Component
Application Software Component
Application Layer
Application Software Component
Application Software Component
Runtime Environment (RTE)
Services Layer
AUTOSAR Operating System (OS)
ECU Abstraction Layer
Complex Drivers
Microcontroller Abstraction Layer
ECU Hardware

Figure 1: Overview of the AUTOSAR layered architecture

The remainder of this paper is organized as follows: The testing and evaluation of real-world implementations of ADAS imposes a certain set of additional requirements, which are discussed in section 2 before giving an overview of the related work. The general concept and architecture of our emulation approach are described in section 3. In section 4 we evaluate the performance and scalability of our approach by means of an exemplary scenario and discuss its benefits and limitations. Section 5 concludes the paper and gives an outlook of future work.

2. BACKGROUND AND RELATED WORK

Before we proceed to the discussion of related work, it is essential to illustrate our scope and area of application as well as the resulting requirements. In order to evaluate and validate real implementations of ADAS in a simulated, virtual environment, a holistic view of the vehicular ad hoc network comprising the three domains vehicle, network and application is necessary. In contrast to existing approaches we aim to not only cover the network characteristics but also the behavior of the vehicles and the network-aware applications in high fidelity. A high fidelity representation of an application means that the actual code as well as the context in which it is executing must be integrated into the overall simulation.

Unlike traditional PC-based software, driving assistance systems are typically executed on embedded hardware platforms and must comply with hard real time requirements. A specific software architecture called AUTOSAR was developed by the automotive industry for this specific purpose, which defines a generalized architecture for Electronic Control Units (ECUs). As shown in figure 1 this model features a separation into multiple layers. AUTOSAR also contains the definition of an embedded real time operating system as well as the possibility to define custom interfaces and runtime behavior in a formal description. The different components of an ECU, e.g. Application Software Components (SWC), can be developed separately from each other and are combined to the desired overall functionality later on. A key benefit of AUTOSAR is the standardization of hardware abstraction layers, which enables hardware-independent development and portability of the majority of an ECU's software. We will exploit this hardware abstraction in our emulation framework to provide a realistic execution environment for the evaluation of network-based applications.

In order to state the fundamentals of this investigation we give a brief overview of existing approaches for VANET simulation and network emulation in the following sections.

2.1 VANET simulation

The usual strategy to simulate VANETs found in literature is to bidirectionally couple a network simulator and a microscopic traffic simulation. Following this approach the interactions between road traffic and network protocols are represented and the mutual impact can be explored [15, 12].

A number of VANET research simulation frameworks which employ this coupling strategy have been developed. They allow researchers to focus on their specific area of interest, i.e. low-level networking such as medium access or high-level concepts of applications such as lowering CO2-emissions or reducing traffic jams. In Veins [15] the application behavior is directly incorporated into the network simulator as a high-level and simplistic model. While VSimRTI [13] and iTETRIS [11] provide specific interfaces for integrating VANET applications into the simulation context, adapting real-world implementations of automotive embedded software to these interfaces requires code modifications.

Since large-scale simulations are usually conducted to perform a statistical analysis of the simulation results, efficient but rather simplistic microscopic traffic simulators are used to generate realistic mobility models. When testing and evaluating real ADAS implementations a more detailed representation of a vehicle's state including its sensors and actuators in the simulation is absolutely vital.

2.2 Network Emulation

Network simulation and real world testbeds are the usual methodologies for evaluating network protocols and applications. Due to the simplifications performed in simulators regarding application models as well as the costs and insufficient repeatability in testbeds, it is desirable to combine the strengths of both methodologies in a network emulator.

The original definition of network emulation by Fall [5] covers the real time coupling of a discrete event network simulator and hardware executing real implementations of software prototypes. In a wider sense, network emulation can be defined as a hybrid experiment technique that combines both real and simulated network components with *real* referring to either hardware or software components [1, p. 14].

When the network simulation can not be executed fast enough due to complex models and high node counts, simulation overload causes the network simulator to lag behind the real time execution of the software prototypes and thus invalidates the results of the network emulation [19]. Since computational resources can usually not be increased infinitely to speed up the network simulation, several attempts have been made to slow down the execution of the real world implementations to match the execution speed of the network simulation. A common approach is to exploit virtualization to decouple the time perception of the software prototype from the wall clock time [19, 16]. The run-time behavior of such a virtualized system is under full control, so it can be synchronized with the network simulation in *virtual time*. Network emulation based on virtualized PC operating systems such as Linux is widely established for evaluating PC-based software, however this methodology is yet novel in the research area of automotive embedded systems and inter-vehicular networks.

3. EMULATING VEHICULAR AD HOC NETWORKS

We now present the design and implementation of our emulation framework. After describing the overall concept we explain in detail the four components of which the framework is composed.

3.1 Conceptual overview

Our framework is designed to provide a generic emulation platform for evaluating real implementations of ADAS which are based on vehicular network communication. The main goal is the support of executing *unmodified* applications in a *high-fidelity* and *accurate* representation of the VANET. The underlying concept is based on the feedback coupling of detailed subsystems for each of the relevant domains constituting a VANET, i.e. the physical domain of each vehicle, the logical domain embodied by the applications running on the ECUs as well as the communication network connecting the vehicles through the wireless channel. Figure 2 shows the three relevant domains as well as the data flows between those subsystem representations in a conceptual overview of the emulation framework. For reasons of clarity, the data flow from vehicle simulator to network simulator is not depicted but the node positions in the network simulator are kept consistent with the vehicle simulator.

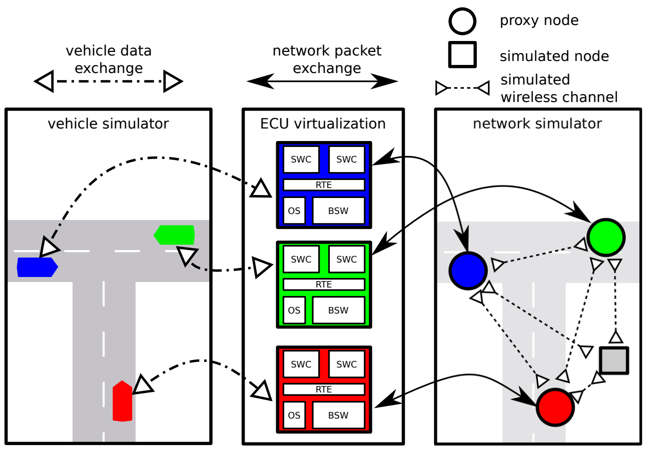

Figure 2: Conceptual overview of the emulation framework

In order to allow evaluation and testing of unmodified applications, the emulation framework needs to provide an execution environment which is as close as possible to the real system on which the applications will be deployed in series production. This could be achieved by representing the logic domain by real hardware ECUs executing the software prototypes. However, this approach is infeasible for the following reasons:

The development process in the automotive industry is characterized by concurrent engineering in order to shorten the time to market. In the given context this especially covers the parallel design and development of both ECU hard- and software, which results in only relatively late availability of the hardware and would thus delay testing of the software prototypes. Additionally, conducting large-scale scenarios would require a large number of ECUs as well as a high logistic effort for setting up and performing the actual experiments. Last but not least, the aforementioned simulator overload resulting from complex models and high node counts in the network simulator can invalidate the evaluation results.

For these reasons we choose to integrate *virtualized* ECUs (VECUs) as the representation of the logical domain into the overall emulation framework. This approach solves the dependency on hardware availability and the scalability issues and allows us to decouple the emulation from the real time constraint by synchronizing the time progression of the software prototypes with the execution speed of the other simulators.

3.2 Network Simulation

The network simulation is used to model the wireless communication network connecting the vehicles and the applications running on their ECUs. As shown in figure 2 each virtual ECU is represented by a proxy node in the network simulation domain. The proxy node acts as a communication endpoint to initiate the simulated transmission of network packets as well as to receive network packets transmitted by other network nodes. Additionally, fully simulated nodes can be included, which may for example represent intelligent infrastructure such as traffic lights.

We apply the *vertical* emulation concept which is defined in [7] and also referred to as a split stack in [14]. The network stack is separated into two parts where the upper layers (including the application layer) belong to the VECUs, while the lower layers are realized by the network simulator. To offer the highest degree of generality and flexibility, the emulation boundary, i.e. the layer at which the network stack is split up, is drawn at the Medium Access Control (MAC) layer. This allows to evaluate arbitrary routing and transport layers as well as the application functionality, which are typically implemented in software and executed in the VECU. Network packets generated by these layers are captured at the virtual Network Interface Controller (vNIC), which is described in the next section. The packets are then injected into the corresponding proxy node, traverse the simulated MAC and physical (PHY) layer and are then potentially received in reverse order at other nodes after the simulated transmission has been performed by the network simulator. The proxy nodes therefore handle all lower layer functionality that is usually performed by hardware. This hybrid emulation approach is shown in figure 3.

In order to enable communication between fully simulated nodes and VECUs above the MAC layer, the fully simulated nodes need to have compliant implementations of the relevant VANET protocols (e.g. routing protocols such as GeoNetworking) and, if necessary and applicable, also application models which can act as traffic sources, e.g. transmitting periodic beacons.

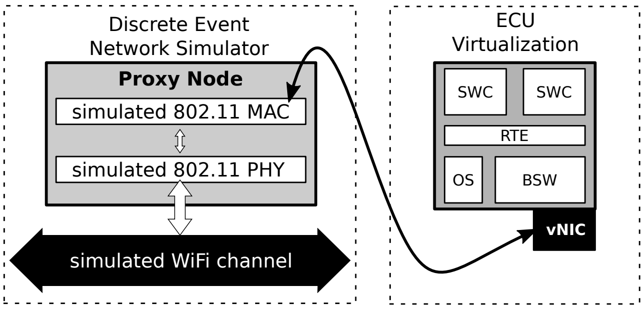

Figure 3: Hybrid Vehicular Ad hoc Network Emulation

The event-driven network simulator ns-3 is chosen to perform the actual network simulation of the wireless communication domain. ns-3 features an open-source modular architecture that can be extended quite easily. A rich number of simulation models is already available in ns-3, of which we employ the WiFi models and specifically the 802.11p MAC layer model [2]. Network packets in ns-3 are represented as binary packets in network byte order that match their real-world counterparts, so it is possible to directly exchange packets between simulation nodes and external systems without the need for any packet translation through the proxy nodes and a custom data-exchange interface.

To allow synchronization of the network simulator with the other domain representations we implemented a custom event scheduler which can be controlled from the outside. In contrast to the default implementation this scheduler executes only those events whose associated simulation time is below a given boundary in virtual time. When this boundary time is reached or a network packet is received by a proxy node, event execution is suspended and the time of the next event in the network simulator's queue is reported to the outside. The synchronization algorithm is described in detail in section 3.5.

3.3 ECU Virtualization

As described in section 3.1 our emulation platform is based on the virtualization of ECUs. While there are various approaches available for virtualizing such embedded systems, the method of choice is justified by two main reasons. The final hardware design of an ECU is usually determined rather late in the development cycle. Therefore important details such as processor architecture, core count etc., which are vital for a detailed modeling of the underlying hardware, are missing until the hardware is specified. Additionally, detailed instruction set or even cycle-accurate simulations require a high computational effort, which conflicts with our goal to conduct large-scale evaluations.

We have thus chosen the rather hardware abstract approach ETAS Virtual ECU[1] which allows us to put the emphasis not on one single, highly-detailed modeled ECU but on the overall system of connected vehicles. This tool enables us to create virtual ECUs based on a formal AUTOSAR architectural model and the hardware independent C code. This approach can be described as host-compiled paravirtualization [3] where the hardware-abstraction layers of AUTOSAR are exploited by porting those abstraction layers as well as an AUTOSAR compliant operating system to a standard PC operating system such as Linux. This allows the execution of a VECU on a traditional desktop PC on top of the host operating system rather than interacting directly with the actual hardware. A VECU is compiled into a self-contained executable that can be instantiated as often as necessary, which enables performing large-scale evaluations. Each VECU is run as a separate process which has its own virtual hardware (e.g. interrupt controller) modeled on an abstract functional level.

[1]http://www.etas.com/en/products/isolar_eve.php

In the following we describe the execution concept of a VECU. The execution is stimulated by an internal clock or through virtual interrupts. The internal clock can either progress with respect to the wall clock when running in real-time mode or clock ticks can be injected from the outside, which allows full control over the execution of the VECU. Due to the fact that the virtual ECU is executed only on a rather abstract hardware model and since the compiler for the host PC is different from that of the target platform, the execution durations of individual tasks are not representative. We thus interpret the execution of VECU tasks as discrete events which means that a task is executed by an infinitely fast processor in terms of simulated time, as time does not progress during the execution of a task. This assumption leads to the fact that preemption of tasks by higher priority tasks or interrupts does not occur, however considering these scheduling effects only makes sense if a more detailed model of the target platform is available.

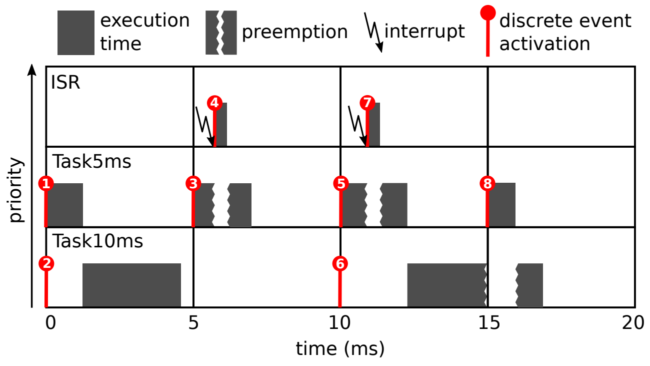

Figure 4: Timing behavior comparison of virtualized ECU

Figure 4 shows the timing behavior during normal execution and when assuming task activations as discrete events by means of an exemplary ECU, which has two cyclic tasks and one Interrupt Service Routine (ISR). During normal execution, tasks can be preempted by tasks of higher priority, which is not accounted for when activating the task execution as discrete events. Task activations and interrupt handling are modeled by the discrete-event execution in the correct order but conclusions about the timing behavior of the target hardware platform cannot be drawn. The duration in terms of virtual time between two clock ticks is arbitrary, so the time resolution is configurable for the desired accuracy. By default this time span is set to 1 ms. The discrete event interpretation allows to achieve deterministic and repeatable evaluation of the software implementation under test because influences stemming from the host operating system do not have an impact on the timing behavior of the virtual system.

A VECU can communicate with the outside world through virtual hardware devices. Since there is yet no standardized integration of VANET hardware in the AUTOSAR architecture we have integrated the virtual wireless network interface (vNIC) using a complex device driver as shown in figure 5. The vNIC redirects the network packets originating from the VECU to the corresponding proxy node in the network simulator. It also offers the interface to inject network packets into the VECU which have arrived at the proxy node. When a packet is injected into the virtual network device, an interrupt in the VECU is raised and in the corresponding ISR the packet can be handled by a custom network stack implementation and the AUTOSAR software components.

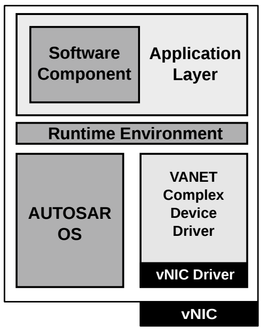

Figure 5: Integration of the virtual network device into the VECU architecture

3.4 Traffic And Vehicle Simulator

As stated in section 2.1 the microscopic vehicle models which are usually employed for conducting VANET simulations are not detailed enough for our purpose. The vehicle dynamics as well as actors and sensors need to be modeled in sufficient detail and accuracy in order to supply all necessary state variables to the real implementation of the ADAS under evaluation.

We therefore employ the nanoscopic traffic and vehicle simulator VIRES Virtual Test Drive (VTD) for the high-fidelity simulation of the physical domain of the vehicles. VTD has been developed for the automotive industry as a virtual test environment used for the development of ADAS [18]. Its focus lies on interactive high-realism simulation of driver behavior, vehicle dynamics and sensors. VTD is highly modular, so any standard component may be exchanged by a custom and potentially more detailed implementation. Its standard driver model is based on the intelligent driver model [17], however an external driver model may be applied if necessary. The same concept applies for the vehicle dynamics simulation, where the standard single-track model can be substituted by a complex vehicle dynamic model adapted for specific vehicles. Each simulated vehicle can be equipped with arbitrary simulated sensors, for example RADAR sensors or synthetic video cameras. VTD offers proprietary interfaces to control the simulation execution in a time driven manner as well as to extract the simulation state after the computation of a simulation step.

3.5 Simulation Synchronizer & Scheduler

The three described domain representations are either time-driven (vehicle simulator), event-driven (network simulator) or both (VECUs). In order to achieve a deterministic co-simulation comprised of all three domains, the subsystems must be synchronized. The Simulation Synchronizer & Scheduler (SSS) ensures that the execution of the subsystems is synchronous so that no time drifts can occur as well as causality errors, i.e. executing events from the past, are avoided. Since none of the system representations allows to perform rollbacks, an optimistic synchronization algorithm cannot be used; we therefore choose a conservative synchronization algorithm.

SSS maintains a global event list to determine which system representation is to be scheduled next. After the execution of a system representation is completed, this system is rescheduled when it is due the next time. The determination of this *next time* depends on the respective system. The vehicle simulator is scheduled once every time step T_{veh} which is configurable for the vehicle simulator. The execution of the VECUs can be scheduled every tick T_{tick} which corresponds to the time resolution of the VECUs as described in section 3.3. In order to reduce synchronization overhead, the task activation behavior of the VECUs, which is contained in the AUTOSAR architectural model, can be exploited. If a VECU's minimum task activation period is $T_{vecu,min}$, this value can serve as the rescheduling period without sacrificing accuracy. The custom scheduler implementation of the network simulator reports its next event time as described in section 3.2 which serves as the next event time in the global event list of SSS for the network simulator. In order to avoid causality errors, the network simulator is only allowed to progress in virtual time until the next VECU will be executed again. This boundary is also derived from the global event list.

We illustrate the synchronization algorithm by the sequence diagram in figure 6, which shows the exemplary scenario of two VECUs which send ping requests and replies over a simulated wireless channel. For reasons of simplicity the vehicle simulator is left out. The transmission durations in this example are purely fictional and listed for demonstrative purposes only. The VECUs in this example exhibit a task activation period of $T_{vecu,min} = T_{tick} = 1.0\,\mathrm{ms}$.

Initially the two VECUs execute one tick of virtual time in parallel. VECU$_2$ sends a ping request at time $t = 1.0\,\mathrm{ms}$ contained in network packet p to VECU$_1$. Packet p is captured at the vNIC of VECU$_2$ and is then sent to SSS which forwards it to the network simulator. This causes the enqueuing of an ns-3 event which injects the packet into the network simulation at $t = 1.0\,\mathrm{ms}$ through the proxy node associated with VECU$_1$. ns-3 is now allowed to execute events until $t = 2.0\,\mathrm{ms}$, which leads to the transmission of p on the simulated wireless channel. At time $t = 1.2\,\mathrm{ms}$ packet p is received at the proxy node, which corresponds to VECU$_1$. This suspends the execution of events inside the network simulator and packet p is delivered through the SSS to vNIC of VECU$_1$, which triggers an interrupt. The corresponding ISR handles the packet in the network stack and sends a ping reply in a response packet r which travels the exact opposite way back to VECU$_2$.

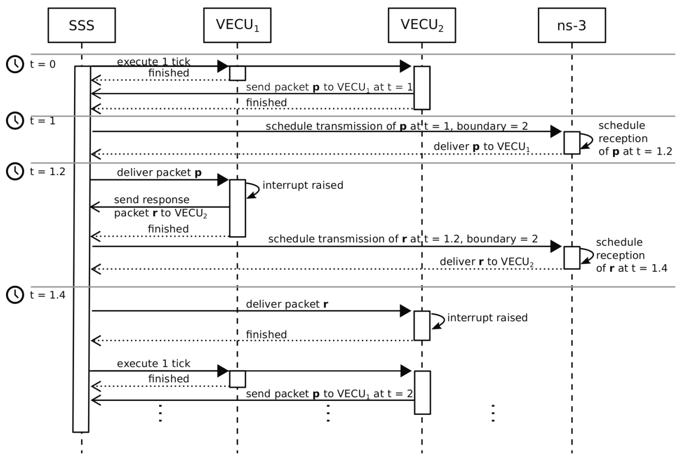
SSS
VECU$_1$
VECU$_2$
ns-3
t = 0
execute 1 tick
finished
send packet **p** to VECU$_1$ at t = 1
finished
t = 1
schedule transmission of **p** at t = 1, boundary = 2
schedule reception of **p** at t = 1.2
deliver **p** to VECU$_1$
t = 1.2
deliver packet **p**
interrupt raised
send response packet **r** to VECU$_2$
finished
schedule transmission of **r** at t = 1.2, boundary = 2
schedule reception of **r** at t = 1.4
deliver **r** to VECU$_2$
t = 1.4
deliver packet **r**
interrupt raised
finished
execute 1 tick
finished
send packet **p** to VECU$_1$ at t = 2

Figure 6: Synchronization of virtual ECUs and network simulation

In our implementation, the system representations do not communicate directly with each other but through the SSS. The underlying federation concept is derived from the High Level Architecture (HLA), a generic framework for distributed simulations [9]. Each system representation is connected to the SSS by means of a specific ambassador software component which is responsible for message exchange in both directions as shown in figure 7. These messages involve both the synchronization and the exchange of simulation state data as depicted in figure 2. The ambassadors translate the messages from SSS to the respective subsystem and vice versa. This allows to replace any given subsystem by either another software implementation or even by real hardware by modifying just the corresponding ambassador. The flexibility of the architecture also makes it possible to add additional simulators to the overall simulation and to distribute the system representations on multiple machines.

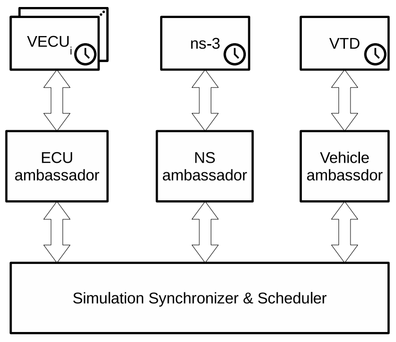

Figure 7: Implementational overview of the emulation framework

4. EVALUATION AND DISCUSSION

In the following we evaluate the proposed framework by means of a synthetic scenario to examine the timing accuracy and performance with regard to scalability. We then discuss the universal applicability of our approach as well as its limitations.

4.1 Evaluation Scenario and Setup

The evaluation is performed by comparing the framework's results and performance with pure network simulation. In order to achieve a good comparability we chose to use the previously described ping scenario with static node positions. In this scenario there are $n = 2k$ nodes where each node $i \in [1, k]$ pings another node $j \in [k + 1, n]$ over a simulated 802.11p Wifi channel with an interval v between requests. We integrated the open source IP stack lwip [4] (version 1.4.1) in the AUTOSAR VECU, which is straightforward due to it being implemented in C. All experiments were carried out on a single machine equipped with an 3.6 GHz Intel Xeon CPU, 16 GB RAM on a 64 bit Linux 3.16 kernel using ns-3 version 3.22 and ETAS Isolar-EVE version 2.2.

4.2 Accuracy and Scalability

To validate the correct synchronization behavior of our global event scheduler we performed the above described scenario once in ns-3 alone without the SSS and any VECUs being attached, so all nodes were fully simulated. The same scenario was then run in our emulation framework with each node now configured as a proxy attached to a VECU instance. The transmissions were simulated on a 802.11p wireless channel at 5.9 GHz. The node count was set to $n = 20$ and the ping interval to $v = 10$ ms. Figure 8 shows an excerpt of the captured round trip times resulting from both the simulation and the emulation experiment between a corresponding pair of nodes. The round trip times vary due to the interference of competing transmissions between the other node pairs on the shared wireless medium. The resulting round trip times are identical for both experiments, which demonstrates that the scheduling of VECUs and network simulation in our emulation framework is performed correctly and deterministically.

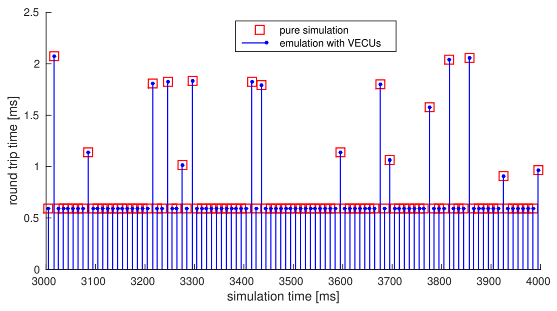

Figure 8: Round trip times in simulation and emulation

Before comparing performance of simulation and emulation we ran the scenario with network simulation disabled. This setup allows to examine the influence of the synchronization tightness by either scheduling the VECUs every $T_{tick} = 1$ ms or every $T_{vecu,min} = 10$ ms. Figure 9 shows the impact of the different synchronization periods. While 40 VECUs can be run synchronously in real time in our framework when using the 10 ms period, the synchronization overhead of the 1 ms period is clearly visible and the real-time boundary is crossed when executing more than 24 VECUs.

In order to quantify the computational overhead in compar-

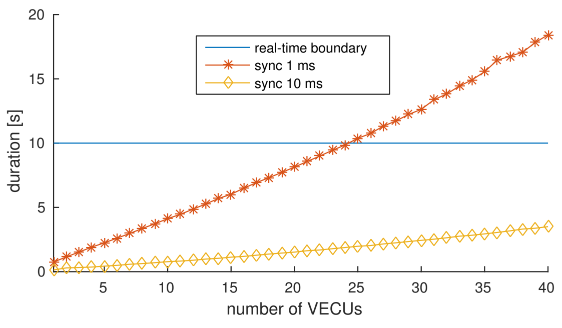

Figure 9: Performance comparison of VECU synchronization periods

ison with the network simulation, which is introduced by our framework, we conducted a series of experiments with increasing node counts for both the simulation alone and the emulation and measured the real-time duration for each configuration. Each configuration was run 10 times for a duration of 10 s of simulated time and the number of node pairs k was increased from 1 to 24. Figure 10 shows the durations for the simulation in ns-3 alone as well as for the emulation with the two different synchronization periods of the VECUs when running the ping scenario with an interval of $v = 100$ ms between ping requests. This results in a message frequency of 10 Hz, which is typical for cooperative awareness in VANET applications [8, p. 275].

The pure simulation obviously performs fastest since no additional synchronization is necessary. The synchronization overhead, which the emulation results exhibit, stems from the fact that the VECUs and the network simulation are executed sequentially when sending and receiving network packets to guarantee a correct and deterministic co-simulation.

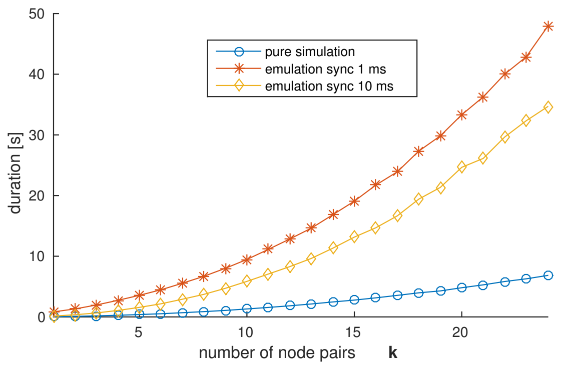

Figure 10: Performance comparison between simulation and emulation

The computational effort which is necessary to perform the network emulation depends on multiple factors. The synchronization period of the VECUs only shows a rather slight impact on the overall duration, whereas the number of nodes represented by VECUs as well as the amount of messages transmitted and received by VECUs affect the performance the most. Another factor, which is not examined here, is the actual workload of each VECU. If the ADAS under evaluation performs complex calculations, this will lead to an additional increase of computational requirements.

4.3 Discussion

The above shown evaluations state that our framework allows to accurately evaluate and test real-world implementations of VANET applications. The overhead introduced by executing VECUs for each node as well as synchronizing the three subsystem representations leads to longer simulation durations. However since the simulation is decoupled from the real-time constraint, the accuracy of the results is not affected even when conducting large-scale experiments.

In the following we will discuss the universal applicability of the approach as well as its limitations.

The chosen virtualization method generally allows to integrate unmodified source code of the ADAS implementations. However, due to the paravirtualization it is necessary to compile the code to run on the x86 host architecture. Typically, software components developed for the AUTOSAR architecture are written in ANSI C. Due to the AUTOSAR hardware abstraction layers, they are independent from the underlying hardware which allows to re-compile the code for the x86 architecture without code modification. However, if the standardized interfaces of the AUTOSAR architecture are bypassed somehow, the source code may need to be modified. Additionally, if source code is not available, e.g. due to IP protection, closed source components can be integrated as precompiled x86 libraries.

While we focus on ad hoc communication based on IEEE 802.11p, our concept is agnostic to the underlying network topology and transmission medium. This allows to evaluate the network-based ADAS on other radio technologies such as ad hoc LTE only by changing the configuration of the network simulator to apply other simulation models for the PHY and MAC layer. In our examples we only consider one ECU per vehicle, however the approach is flexible enough to allow the integration of multiple ECUs per vehicle and, given that suitable models exist, even the simulation of intra-car networks such as CAN.

Since the chosen virtualization approach assumes no detailed knowledge of the target hardware, hardware characteristics which might influence the timing behavior are not taken into consideration. Delays which are caused by the target hardware are neglected, which is a limitation of our current implementation. We regard the modeling of hardware-introduced delays as future work which can be approached by instrumenting and tracing execution on real hardware platforms once they are available in the development process [10].

5. CONCLUSION

In this paper we proposed a novel approach for emulating vehicular ad hoc networks for evaluation and testing of network-aware automotive embedded systems. The presented methodology employs virtualization to allow the integration of real-world automotive software into the overall simulation consisting of the three coupled subsystem representations of the physical, application logic and wireless networking domain. This enables the detailed analysis of network protocol and application implementations in the context of a realistic runtime execution provided by an AUTOSAR compliant embedded operating system. A global simulation scheduler synchronizes the execution of all domain representations to achieve a deterministic and correct experiment execution. The evaluation shows that the emulation generates accurate results by synchronously executing the software components encapsulated in virtual ECU instances and the other simulators. The approach allows to perform detailed and large-scale evaluations early in the product development cycle without being dependent on the availability of real hardware.

As our next steps we plan to integrate an industry implementation of a Car2Car communication stack into our proposed virtual prototype solution as well as tackle the area of hardware-in-the-loop simulation by combining both real and virtual ECUs.

6. REFERENCES

[1] R. Beuran. *Introduction to Network Emulation.* Pan Stanford Publishing, 1st edition, 2012.

[2] J. Bu, G. Tan, N. Ding, M. Liu, and C. Son. Implementation and Evaluation of WAVE 1609.4/802.11P in Ns-3. In *Proceedings of the 2014 Workshop on Ns-3*, WNS3 '14, pages 1:1–1:8, New York, USA, 2014. ACM.

[3] J.-L. Béchennec, M. Briday, S. Faucou, F. Pavin, and F. Juif. ViPER: A Lightweight Approach to the Simulation of Distributed and Embedded Software. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, 2010.

[4] A. Dunkels. Design and implementation of the lwip tcp/ip stack. Technical report, Swedish Institute of Computer Science, 2001.

[5] K. Fall. Network emulation in the vint/ns simulator. In *Proceedings of the fourth IEEE Symposium on Computers and Communications*, pages 244–250, 1999.

[6] O. Gietelink, J. Ploeg, B. De Schutter, and M. Verhaegen. Development of advanced driver assistance systems with vehicle hardware-in-the-loop simulations. *Vehicle System Dynamics*, 44(7):569–590, 2006.

[7] E. Göktürk. Emulating ad hoc networks: differences from simulations and emulation specific problems. In *New Trends in Computer Networks*, volume 1 of *Advances in Computer Science and Engineering: Reports*. Imperial College Press, October 2005.

[8] H. Hartenstein and K. Laberteaux. *VANET Vehicular Applications and Inter-Networking Technologies.* Intelligent Transport Systems. Wiley, 1. edition, 2010.

[9] IEEE. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules, August 2010.

[10] S. Kristiansen, T. Plagemann, and V. Goebel. Modeling communication software execution for accurate simulation of distributed systems. In *Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '13, pages 67–78, New York, USA, 2013. ACM.

[11] M. Rondinone, J. Maneros, D. Krajzewicz, R. Bauza, P. Cataldi, F. Hrizi, J. Gozalvez, V. Kumar, M. Röckl, L. Lin, et al. iTETRIS: a modular simulation platform for the large scale evaluation of cooperative ITS applications. *Simulation Modelling Practice and Theory*, 34:99–125, 2013.

[12] F. J. Ros, J. A. Martinez, and P. M. Ruiz. A survey on modeling and simulation of vehicular networks: Communications, mobility, and tools. *Computer Communications*, 43:1–15, 2014.

[13] B. Schünemann. V2X simulation runtime infrastructure VSimRTI: An assessment tool to design smart traffic management systems. *Computer Networks*, 55(14):3189 – 3198, 2011.

[14] C. Serban, A. Poylisher, and J. Chiang. Virtual ad hoc network testbeds for network-aware applications. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 432–439, Osaka, Japan, 2010. IEEE.

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

[16] F. Sultan, A. Poylisher, J. Lee, C. Serban, C. J. Chiang, R. Chadha, K. Whittaker, C. Scilla, and S. Ali. Timesync: Enabling scalable, high-fidelity hybrid network emulation. In *Proceedings of the 15th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '12, pages 185–194, New York, USA, 2012. ACM.

[17] M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805, 2000.

[18] K. von Neumann-Cosel, M. Dupuis, and C. Weiss. Virtual test drive - provision of a consistent tool-set for [D,H,S,V]-in-the-loop. In *Proceedings of the Driving Simulation Conference*, Monaco, 2009.

[19] E. Weingärtner, F. Schmidt, H. Vom Lehn, T. Heer, and K. Wehrle. Slicetime: A platform for scalable and accurate network emulation. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)*, Boston, USA, March 2011.