Simulator Coupling for Network Fault Injection Testing

Emilia Cioroaica Thomas Kuhn

Embedded Software Engineering, Fraunhofer IESE, Germany,
{Emilia.Cioroaica,Thomas.Kuhn}@iese.fraunhofer.de

Abstract

System architectures of embedded systems are undergoing major changes. Embedded systems are becoming cyber-physical systems (CPS) with open interfaces and resulting distributed control loops. This calls for new testing approaches that enable early evaluation of system and safety concepts, and support the evaluation of system designs before they are implemented. Simulation is a common technology that supports the testing of embedded systems, but existing simulators are focused and specialized. A single simulator often does not support all models needed to provide a valid testing environment for system designs. In this paper, we describe our framework for the coupling of communication simulators to enable virtual testing and safeguarding of embedded system designs. The integration of network simulation models and fault injectors enables testing of safety concepts. The applicability of our approach is illustrated in the context of a case study based on a vehicle system design realized as contract work.

Keywords: virtual testing, simulation, simulator coupling, communicating systems, embedded systems, fault injection

1 Introduction

Enabling the development of safety concepts for open systems is one of the most pressing challenges in embedded systems development. Formerly self-contained and isolated systems are being equipped with open interfaces to enable communication. Closed safety-relevant control loops open up and integrate remote devices via wireless networks. This requires the development of new safety concepts that include detection and handling of potential transmission errors. The development of safety concepts becomes even more challenging when wireless links are used, because these are much less predictable than wired links.

Development and testing of communication systems has always been supported by simulations. They provide a virtual testbed that enables the evaluation of application and protocol aspects. Safety concepts could benefit from this technology as well. However, network simulators are often optimized with respect to performance simulation. The provided protocol and network models accurately resemble timing; errors are, however, only represented by an error flag indicating the presence of a transmission error, and this causes, for example, the dropping of a faulty frame. For the development of safety concepts, a more detailed simulation of faults is necessary. The loss of a frame, for example, is not significant from the viewpoint of a safety engineer. Much more significant are flipped bits, which lead to wrong data being received that is not detected by CRC checksums. Safety concepts need to handle these faults as well. Fault injection testing (Guthoff and Sieh, 1995; Barton et al., 1990; Zussa et al., 2014) is one approach for validating designs with respect to their robustness against faults. However, it only covers functional transmission models that support fault injection, but not performance evaluation.

The development of networked embedded systems requires consideration of both aspects: functional transmission of errors and performance optimization. Safety concepts create additional overhead with their measures: they do not only need to be robust against faults and economical with respect to the additionally used resources, but their failure-handling strategies must not affect other communications beyond a permitted degree either. To support the development of next-generation safety concepts, we have therefore developed a framework for coupling fault injection testing with network simulation. In this publication, we document the integration of different network simulators into our simulation framework FERAL (Framework for Evaluation on Requirements and Architecture Level)(Kuhn et al., 2013), and discuss the integration of fault injection testing for the evaluation of safety concepts for open embedded systems.

The remainder of this paper is structured as follows: Section 2 contains a survey of related work. Section 3 presents the challenges of virtual testing on an open embedded system example. Section 4 documents our framework for the coupling of simulation models for network simulation. Section 5 adds the aspect of fault injection testing. Section 6 presents a case study showing use cases of our virtual testing environment. In Section 7, we draw conclusions and lay out future work.

2 Related Work

Most existing approaches for simulator coupling are tailored couplings between simulators to support testing of systems or to predict the properties of a product under development. This leads to considerable overhead because event detection, simulation accuracy, and the correct coupling of execution models must be considered individually for each coupling. The following references indicate the concrete necessity for simulator coupling.

The work presented in (Siddique et al., 2007) illustrates the networking domain as another application area of simulator coupling. Through the layered approach used for most communication stacks (Schumacher et al., 2009), this domain is predestined for the integration of simulators. By coupling simulators for link layer protocols and network layers, the authors of (Siddique et al., 2007) build an infrastructure for locating interactions between effects on both layers. The work described in (Schumacher et al., 2009) applies the simulator coupling approach to the automotive domain to simulate the impact of Car-to-X systems. This requires coupling of the OMNeT++ simulator, which provides a network simulation, and the road traffic simulator SUMO. The coupling has to integrate the position data for each vehicle as created by SUMO with the OMNeT++ simulator, so that wireless networking characteristics and the impact of car movements are correctly simulated, including the effects of the Car-to-X application under evaluation.

PicSim (Björkbom et al., 2011) is a tool for the investigation of networked and wireless control systems where two tools, Simulink and ns-2, run on two different PCs, which are connected via LAN. One of the PCs runs a Simulink model in a MATLAB version either for Windows or for Linux. The second PC runs the network simulation and needs to have a Linux operating system. The two simulators are then started at the same time and are synchronized to share their results. In this way, two important parts of the system - functional and network behavior - can be brought together and interdependencies can be tested. The network behavior focuses on wireless communication.

The approach described in (Zhang et al., 2013) couples the notations/tools CarSim, SystemC, and C-Code to simulate and test the behavior of a cyber-physical system. Such a CPS is made up of three parts: a physical layer, a network/platform layer, and a software layer. CarSim is used to simulate the physical layer, while SystemC handles the network/platform layer and the software layer is described by C code. The framework is enhanced by a tool for model-based design to allow the creation of rapid prototypes. The authors of (Eyisi et al., 2012) present the Networked Control Systems Wind Tunnel (NCSWT), an integrated modeling and simulation tool for the evaluation of Networked Control Systems (NCS). NCSWT integrates Matlab/Simulink and the network simulator ns-2 for modeling and simulation of NCS using the High Level Architecture (HLA) (Kuhl et al., 1999) standard.

Existing fault injection approaches focus on the injection of specific faults or classes of faults at the system level (Barton et al., 1990; Guthoff and Sieh, 1995) or at the software level (Duraes and Madeira, 2006) in order to validate the system's functional behavior.

Related work shows that simulator coupling is a commonly applied technique for evaluating communicating systems. Performance evaluation is a proven technique for predicting network performance. Safety-relevant systems additionally need to be evaluated with respect to their resilience. Consequently, fault injection techniques should be additionally used to evaluate network performance, additional overhead due to safety measures, and the impact of faults in one integrated simulation.

3 Virtual Testing of Open embedded Systems

Figure 1 illustrates the functional structure of a remotely controlled lift system, which will be used as an explanatory example for this paper. This system enables remote control of a hydraulic lift with a smartphone. The development of such a system starts with requirements and high-level models for functional behavior and safety concepts. Simulator coupling, as documented in (Kuhn et al., 2013), enables the coupling of these models into one integrated simulation, and therefore the rapid evaluation of concepts and early feedback to developers.

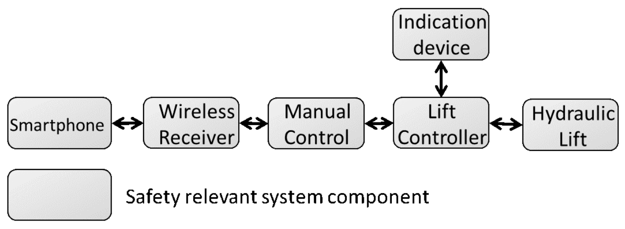


Figure 1. Example Remote Lift Controller System.

For the evaluation of the safety concepts of the example system, a functional structure as shown in 1 is not sufficient. Networks are an integral part of this system, and they affect its safe operation. Safety concepts need to handle network errors reliably. Figure 3 illustrates a revised version of 1 that includes network simulation for the evaluation of safety concepts. Simulation of wireless (LAN) networks is provided by the ns-3 network simulator 0; other simulators are integrated to simulate CAN bus networks and generic wired links. The coupled simulation models split the system into three semantic domains that execute discrete event and discrete time models.

Fault injection testing enables test-based evaluation of safety concepts. To support the evaluation of safety concepts for networked systems, fault injection testing needs to be supported for all network simulation components.

In the context of the example system shown in Figure 3, this includes the ns-3 network simulator, the CAN bus simulation, and the tailored models wired links.

To assure safety, the system behavior needs to be tested in holistic simulations. Faults are injected at the level of simulation components, while behavior needs to be observed both at the component and at the system level. To enable fault injection testing for system designs with different levels of abstraction, fault injection needs to be decoupled from the network simulation.

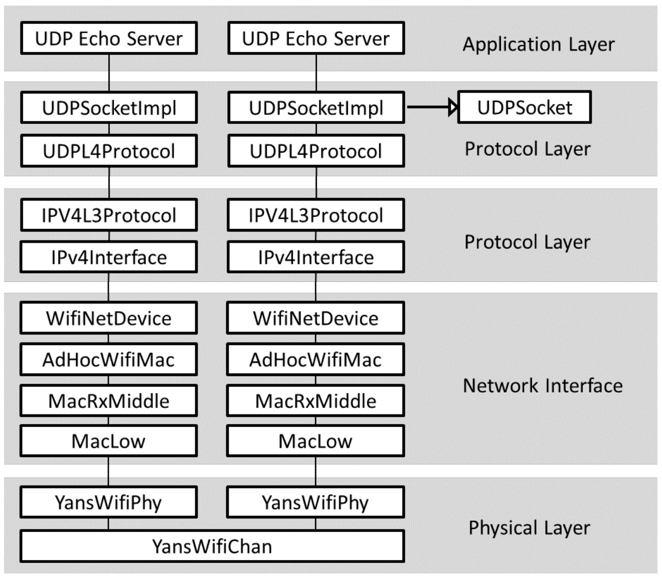


Figure 2. Internal structure of an ns-3 Wi-Fi simulation.

4 Simulator Coupling for Network Simulation

Network simulators offer their simulation models with proprietary interfaces. Development of simulated protocol stacks is possible by combining these interfaces. Figure 4 illustrates the structure of a CAN bus simulation. It consists of the CAN medium, which simulates the CAN bus medium and signal propagation, the CAN bus controllers (CANCtrl), and interface components (CanIF). Interface components aggregate raw data into CAN frames and add offsets, if necessary. On top of the CAN interface components, applications and higher-level protocols like ISOBUS may be implemented.

Developing a generic approach for the coupling of network simulation models and fault injection testing requires an understanding of the inner structure of network simulators, and development of a high-level architecture that encapsulates the components that are specific for each network simulation. Figure 2 and Figure 5 illustrate the architecture of simulated communication stacks for the ns-3 and OMNeT++ simulators.

The example structure shown in Figure 2 illustrates the component instances that create a simulated Wi-Fi network in the ns-3 simulator. Four types of layers can be identified: The Application layer consists of the simulated application behavior. The Protocol layers implement the UDP and IP protocols. The Network Interface layer simulates the medium access control of the Wi-Fi network; its simulation is split into different components. The Physical layer is simulated by components that simulate physical layer encoding and the physical channel. Since ns-3 is a discrete event simulator, frame propagation is based on discrete events as well. Every frame is marked by an event that indicates the transmission of its first bit as well as the frame length. When receiving a physical transmission, every frame is represented by the time that its first bit is received and by the time when its last bit is received. This information is used together with the received signal strength, which is calculated by the propagation model of the physical channel model for the simulation of frame collisions. Upper layers represent frames transmitted and received by one event only. The internal structure of an OMNeT++ simulation is similar. Figure 5 illustrates the component instances that create an Ethernet simulation. It consists of the same basic layers as the ns-3 network simulation. And similar to ns-3, the simulation of the physical channel (EtherBus or YansWiFiChan) is instantiated once; the other components in the Physical layer, the Network Interface layer, and the Protocol layers are instantiated once per simulated network node. Components on the application level may be instantiated multiple times.

The coupling of simulators should yield a protocol stack structure as shown in Figure 4. Applications and application-level protocols should be able to connect to simulated networks. The type of simulated network should be easily replaceable, enabling the simulation of a scenario with both an idealistic network for functional evaluation and its evaluation with a realistic network simulation for performance evaluation.

Figure 6 illustrates our common structure for the integration of network simulation models and functional models. It also shows the types of layers from integrated network simulators that are encapsulated and integrated as simulation components (Kuhn et al., 2013). White-box network simulation models that define all relevant components like queue, medium access control, and application interfaces can be integrated in the same way as black-box components that hide their internal structure. This enables both the rapid integration of existing simulators with low effort as a black-box simulation and the more effort-consuming integration of simulators as white-box components. White-box components require the development of more and additional components, but also enable much finer-grained customization of simulated networks.The component named Application Interface realizes the interface between higher-level protocols and applications, and the network simulation. Its structure is documented in Figure 7.

Applications create PDUs that transmit serialized information through communication networks. In addition to the transmitted data, addressing information might be provided to the simulation components. This addressing information may contain, for example, a CAN bus message ID, or a UDP address including a receiver port number. Adapters may use and change this information to simulate mappings that are realized by the protocol under development. As shown in Figure 7, PDUs additionally contain a list of key/value pairs that store data that is specific to the simulation models and a list of failure modes for this frame.

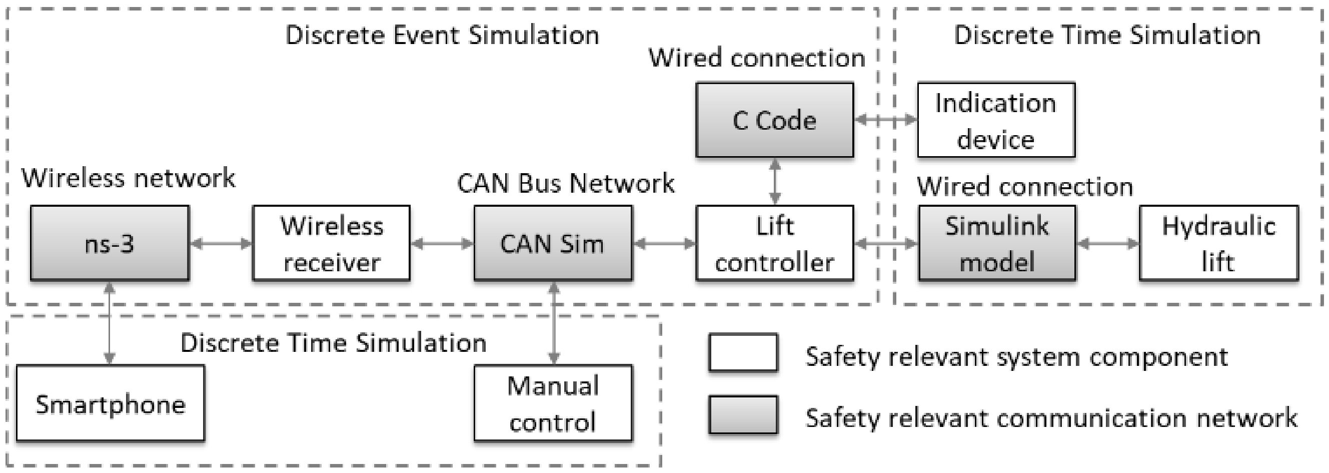


Figure 3. Component types and models of computation of the system example.

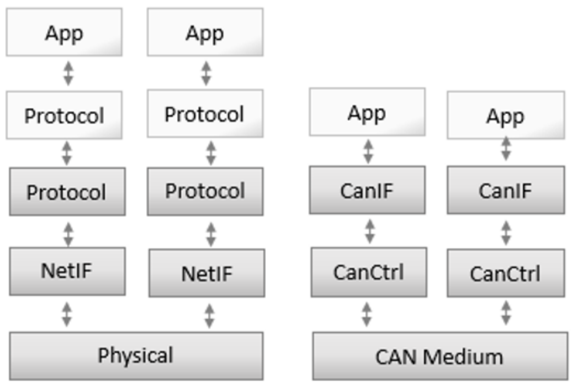


Figure 4. Abstract structure of network simulator.

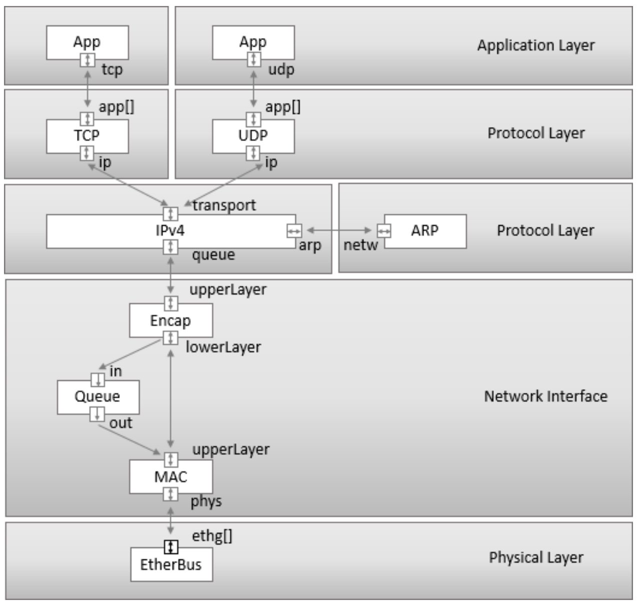


Figure 5. OMNeT++ detailed structure.

5 Network Fault Injection Testing

The integration of network simulators as simulation components enables accurate simulation of networked embedded system components. FERAL enables the rapid development of simulation components and their coupling. Fault injection testing, however, requires a generic approach that is independent of the simulation components that provide the network simulation. Figure 8 lists the relevant failure modes, which were derived from ISO 26262 (ISO 2011) for communication in road vehicles.

While all communication networks basically introduce the possibility of all types of communication failures, the concrete probability of a particular failure depends on the type of communication system to be used. Tailored fault detection and containment mechanisms need to reduce the probability and/or the impact of faults and at the same time conserve resources of the communication system. They add, for example, additional checksums and drop faulty frames instead of passing them to higher protocol layers, preventing the processing of corrupted data by safety-relevant applications.

To effectively simulate failure propagation in embedded systems, the simulation of faults happening and their consequences need to be split from each other. A possible fault on a communication medium is the flipping of multiple bits. A consequence of this fault could be either the dropping of the frame if the bit flip is detected, or a wrong value that is passed to the application.

Our fault injection framework realizes this separation by defining explicit fault injectors and fault processors. Fault injectors create faults in the system. Fault processors simulate consequences of faults. Both fault injectors and fault processors may operate statistically in performance evaluation scenarios to determine performance impacts, or deterministically to evaluate the suitability of a safety concept for the virtual certification of a system.

Figure 9 illustrates fault injection testing with one black-box network simulation component. Fault injectors and fault processors are placed in the communication paths of the network simulation. Both fault injectors and fault processors are added to inbound commu-

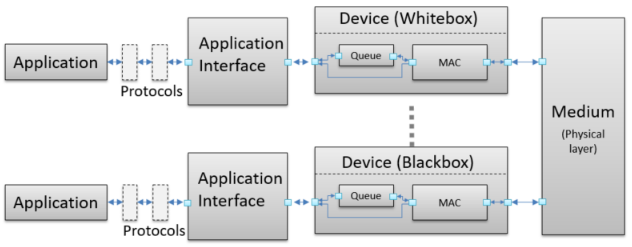


Figure 6. Common structure for integrating network simulation models.

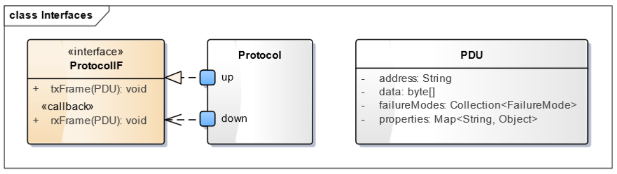


Figure 7. Interface for applications and protocols and PDU structure.

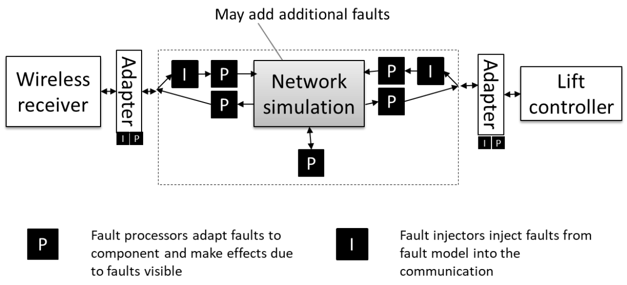

May add additional faults
Wireless receiver
Adapter
I P
I
P
P
Network simulation
P
I
P
P
Adapter
I P
Lift controller
P
Fault processors adapt faults to component and make effects due to faults visible
I
Fault injectors inject faults from fault model into the communication

Figure 9. Fault injectors and fault processors.

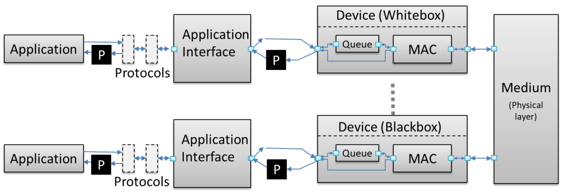
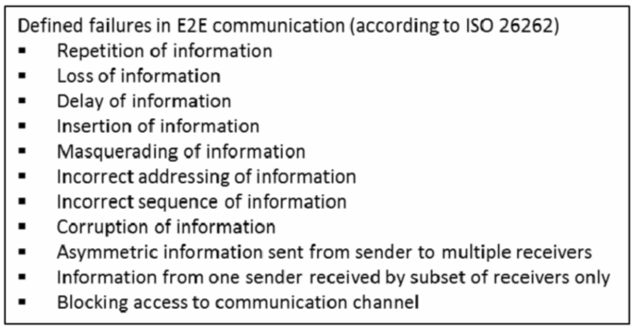


Figure 10. Fault injectors and fault processors.

nication paths, whereas only fault processors are added to outbound communication paths. Fault processors in inbound communication paths therefore can control whether adapters will process the faults for the native simulation model or not. Faults that are created by fault injector components are added to the failure-modes list of transmitted PDUs (cf. Figure 7). Fault processors scan the list of faults for each PDU and modify the PDU based on their implementation.

Fault processors and fault injectors enable fault injection testing independent of network simulation components, and consequently their independent development. Therefore, it is best if network simulation models do not realize fault processing at all, but leave the simulation of faults to explicit fault injector and fault processor components.

Figure 10 illustrates fault injection testing with our common structure for the coupling of functional and network simulation models. Fault injectors on medium inputs inject faults that affect all receivers, e.g. interferences from other transmissions or a broken cable. Injected faults are stored in the failure-modes field of transmitted PDUs, as shown in Figure 7. Fault processors on medium inputs convert injected faults into failures that affect all receivers. Interferences may, for example, lead to shifted bits or dropped frames, or may have no consequence at all if the interference was not sufficiently strong. Fault injectors and processors with medium outputs model effects that only one or a subset of all receivers suffer from. Application-level fault processors enable developers to convert network faults into application-specific faults. For example, they turn undetected bit shifts into application-specific changed inputs. This enables worst-case analysis with critical inputs for application components.



Defined failures in E2E communication (according to ISO 26262)

- Repetition of information
- Loss of information
- Delay of information
- Insertion of information
- Masquerading of information
- Incorrect addressing of information
- Incorrect sequence of information
- Corruption of information
- Asymmetric information sent from sender to multiple receivers
- Information from one sender received by subset of receivers only
- Blocking access to communication channel

Figure 8. Failures in end-to-end communication.

Currently, the fault injectors and processors are available for use at the medium, device, and application level, as in Table 1.

Injectors	Processors
Interference (once)	(Multi) bit change
Interference burts	Frame drop
Signal fading	Frame creation
Cable break	Retransmission
Collision	Delay
Logic error	Sequence change
	Value change

Table 1. Fault Injectors and Processors

Fault injectors and processors support different activation patterns. They can be activated once at a given point in time, in intervals, or sporadically. Fault processors represent both high-level effects caused by protocols, e.g., duplication or the change of frame sequences due to retransmission, and low-level effects like bit changes. At the application level, remaining (uncaught) faults of the interference type may yield a value change that changes frame contents at the logic level.

6 Case Study

The lift remote control example from Section 3 is an anonymized case study based on an industry cooperation project. Horizontal movements of a lift are controlled with a smartphone by at most two operators. Safety measures are applied to ensure that the received user input fully reflects the intention of both users. If contradictory commands are received, movement stops. Periodic heartbeats ensure that communication with both operators is possible. Both the smartphone and the wireless network are considered to be untrusted for system design. Therefore, potential errors in smartphones and networks need to be handled. All safety-relevant processing is performed on the wireless LAN receiver. The case study presented here implements a simulation model that reflects the system model illustrated in Figure 3 with additional fault injection as illustrated in Figure 10. The Wi-Fi network was simulated by the ns-3 simulator; the CAN bus simulation model was developed at Fraunhofer IESE. Corrective actions defined by the safety concept were integrated into the wireless receiver.

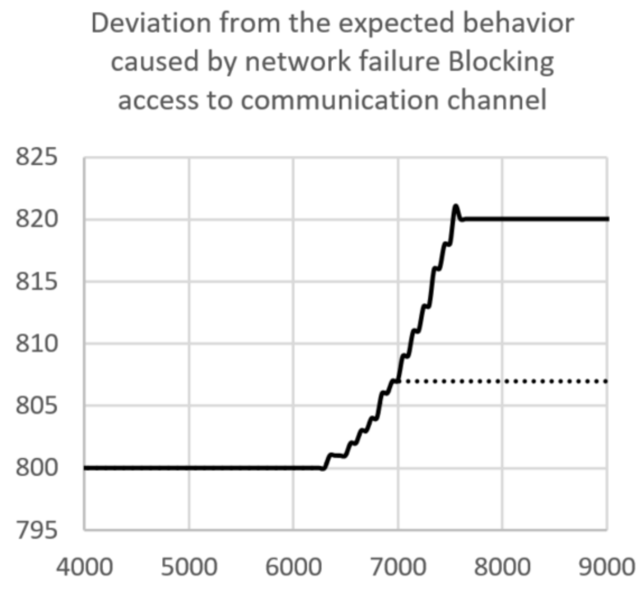


Figure 11. Fault injectors and fault processors.

In this case study, the response of the system to injected faults was evaluated. Once a fault had been injected, it was observed how the system reacted and how well the corrective actions were applied to avoid potential hazards. Figure 11 illustrates one simulated scenario: Due to a simulated network failure, the wireless receiver stops receiving heartbeat messages from the smartphone. As a safety measure, the hydraulic lift stops moving. This is realized by the wireless receiver implementation which, as soon as the heartbeat is not being received anymore, sends stop movement commands to the hydraulic lift. The dotted line represents the user's intended behavior, while the continuous line shows the safe system behavior in the case of the injected fault.

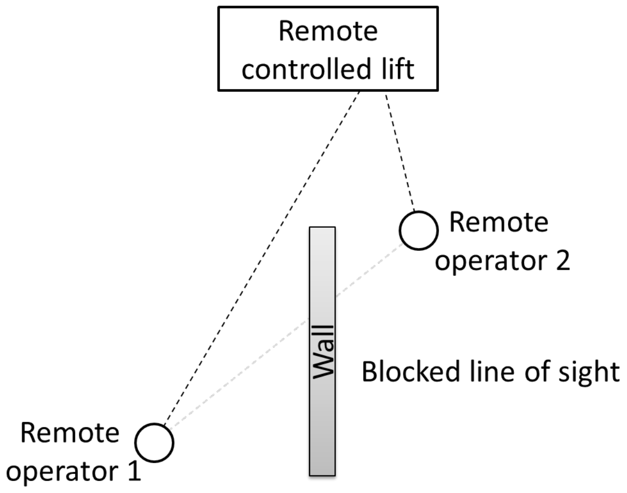


Figure 12. Fault injectors and fault processors.

The second scenario validates another safety function. According to the safety concept, if the two remote operators issue different commands, the lift needs to stop moving immediately. This way, every operator can prevent further movements of the lift if he detects a hazard. Validation of this function requires the integration of a physical network simulation model. As shown in Figure 12, two remote operators are placed at different locations. Due to the topology of the environment, transmissions from the two users to each other are shielded. Therefore, the CSMA mechanism of wireless networks cannot prevent collisions from happening at the wireless receiver. Consequently, control commands collide, which cause the commands with stronger signal strength from operator 2 to be received.

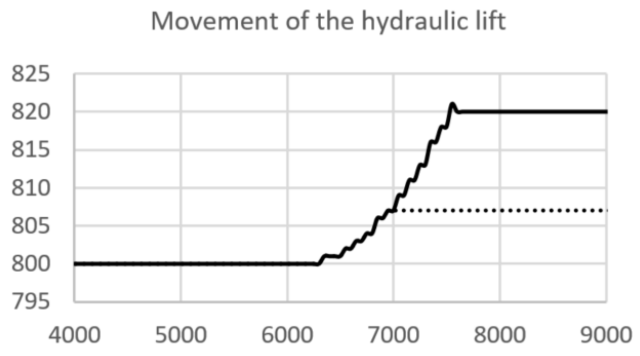


Figure 13. Fault injectors and fault processors.

Frame collisions prevent the more distant operator from communicating with the lift. Due to a sufficient number of heartbeats being properly received, this situation is not detected by the wireless receiver, which is a flaw in the safety concept. As shown in Figure 13, the lift does not stop moving at time 6000, but continues its movement upwards, shown by the solid line. The dotted line represents the expected behavior when two contradictory commands are issued. The simulation results show that this case was not properly handled by the safety concept

7 Conclusions and future work

In this paper, we presented our work regarding simulator coupling for network and functional simulation, as well as simulator-independent fault injection testing. We analyzed existing simulators and devised a reference structure that represents the most important components of communication stacks. Fault injector and fault processor components enable fault injection testing that is independent of the simulators used. Therefore, failure models are independent of functional models and can be used in both functional and performance evaluation simulations.

This enables the rapid development of virtual communication stacks. Integration of fault injection testing using fault injectors and fault processors enables both statistical injection of faults that resemble the error distribution of a real network.

Future work in this area includes the extension of fault injection testing with platform models. Research needs to be done to evaluate whether memory and processor failures as well as failures in different application models can be integrated into simulations using fault injectors and fault processors in a similar manner as that used for integrating communication failures.

References

James H. Barton, Edward W. Czeck, Zary Z Segall, and Daniel P. Siewiorek. Fault injection experiments using fiat. *IEEE Transactions on Computers*, 39(4):575–582, 1990.

Mikael Björkbom, Shekar Nethi, Lasse M Eriksson, and Riku Jäntti. Wireless control system design and co-simulation. *Control Engineering Practice*, 19(9):1075–1086, 2011.

Joao A Duraes and Henrique S Madeira. Emulation of software faults: A field data study and a practical approach. *IEEE transactions on software engineering*, 32(11):849–867, 2006. doi:10.1109/TSE.2006.113.

Emeka Eyisi, Jia Bai, Derek Riley, Jiannian Weng, Wei Yan, Yuan Xue, Xenofon Koutsoukos, and Janos Sztipanovits. Nc-swt: An integrated modeling and simulation tool for networked control systems. *Simulation Modelling Practice and Theory*, 27:90–111, 2012.

Jens Guthoff and Volkmar Sieh. Combining software-implemented and simulation-based fault injection into a single fault injection method. In *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on*, pages 196–206. IEEE, 1995.

Frederick Kuhl, Richard Weatherly, and Judith Dahmann. *Creating computer simulation systems: an introduction to the high level architecture*. Prentice Hall PTR, 1999.

Thomas Kuhn, Thomas Forster, Tobias Braun, and Reinhard Gotzhein. Feral - ̌framework for simulator coupling on requirements and architecture level. In *Formal Methods and Models for Codesign (MEMOCODE), 2013 Eleventh IEEE/ACM International Conference on*, pages 11–22. IEEE, 2013.

Henrik Schumacher, Moritz Schack, and Thomas Kürner. Coupling of simulators for the investigation of car-to-x communication aspects. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 58–63. IEEE, 2009.

Mohammad M Siddique, Andreas J Konsgen, and Carmelita Gorg. Vertical coupling between network simulator and ieee802. 11 based simulator. In *Information and Communication Technology, 2007. ICICT'07. International Conference on*, pages 127–130. IEEE, 2007.

Zhenkai Zhang, Joseph Porter, Emeka Eyisi, Gabor Karsai, Xenofon Koutsoukos, and Janos Sztipanovits. Co-simulation framework for design of time-triggered cyber physical systems. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pages 119–128. ACM, 2013.

Loic Zussa, Jean-Max Dutertre, Jessy Clediere, and Bruno Robisson. Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 130–135. IEEE, 2014.