WARPsim: A Code-Transparent Network Simulator for WARP Devices

Andreas Schumacher, Martin Serror, Christian Dombrowski
Chair of Communication and Distributed Systems
RWTH Aachen University, Germany
{serror, dombrowski}@comsys.rwth-aachen.de

James Gross
School of Electrical Engineering
KTH Royal Institute of Technology, Sweden
james.gross@ee.kth.se

***Abstract*—Analyzing a communication protocol by means of simulation and real-world experimentation requires careful protocol implementation in both domains. Differences in the implementation may lead to significantly diverging performance results, which may affect the protocol design process adversely. A code-transparent simulation and experimentation framework for Wireless Access Research Platform (WARP) devices is proposed, which is called WARPsim. By extending the simulation engine appropriately, the same application code that runs on WARP devices can be used for simulation. This work studies the implications of this approach using the example of implementing time-critical Medium Access Control Layer (MAC) protocols on WARP devices. In the demonstration, various MAC protocols will be simulated using WARPsim, while changing protocol parameters, but also crucial aspects of the emulated hardware. A graphical representation integrated into the framework allows for an intuitive examination of the protocol behavior.**

I. Motivation

Designing and implementing new communication protocols is a challenging task for engineers [1]. After having analyzed and specified the requirements of a communication process, a suitable protocol is designed to carry out this process. Typical tools involve Finite State Machines (FSMs), Unified Modeling Language (UML), Specification and Description Language (SDL) [2], and Message Sequence Charts (MSCs) [3]. Simulating parts of a protocol or the entire protocol is often done prior to prototypical deployments to avoid challenges in manual or automated tests in a potentially highly dynamic environment. Simulations ensure a controllable environment to validate general conformance to the specification, while abstracting from certain real-world details [4], e. g., processing times of an embedded device. Besides providing a controllable environment, larger topologies can be evaluated for which not enough devices may be available. In the next step, a prototypical implementation is derived that allows for the consideration of those previously neglected aspects. Results of simulations and experimentation are compared to the specification. In case of discrepancies, adjustments to the protocol design and implementation are done iteratively.

Transforming the protocol design description into a running implementation, be it for simulation or experimentation, is time-consuming and typically prone to errors as complexity rises. Despite the possibility to generate executable program code from certain descriptions in an automated fashion, e. g., by using SDL or UML, this automatic approach requires additional tools to make rigorous timing statements [5]. Yet, these approaches are constrained by environmental conditions imposed by the devices, especially in case of embedded devices [6]. Expert knowledge and experience is required to tailor a protocol description to a specific implementation platform. If program code used for simulation does not comply to the specification in the same way as the program code of the experimentation does, complex errors may occur. This is in particular true as simulation tools and real-world target devices usually differ quite significantly in terms of programming language, implementation paradigms, and side constraints.

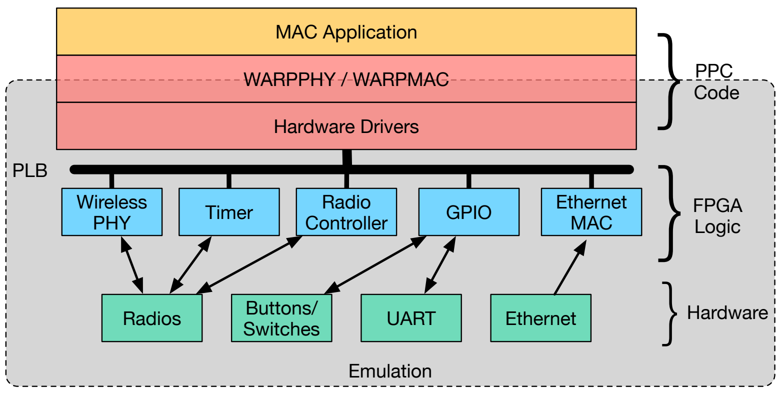
978-1-4799-8461-9/15/$31.00 ©2015 IEEE

In this paper, we investigate a way to reduce the risks of evaluating divergent implementations of the same protocol in simulation and on real hardware by utilizing a code-transparent simulation engine. The goal is to use the same program code that is generated by humans according to the specification to perform evaluation studies in a controlled simulation environment *and* on a real-world experimentation device. Hence, we propose WARPsim, a code-transparent simulator that emulates the underlying hardware of WARP devices [7]. This requires a careful consideration of which device-dependent software or hardware blocks to emulate, and how to match behavior of the emulated blocks to the experimentation device. Emulation is identified to offer huge benefits in the protocol development process [8] as it is an intermediate step between the simulation domain and the real-world. As an example, we consider the development process of time-critical Medium Access Control Layer (MAC) protocols on WARP devices based on the OFDM Reference Design. To the best of the authors knowledge, the simulation and evaluation in a real-world deployment using the same protocol implementation has not gained much attention yet. Various frameworks exist that combine emulation and simulation [8], [9] (and references therein). Related efforts to automatically generate simulation code from formalized descriptions [10] also exist, but they lack the consideration of a code-transparent execution on a real-world device to ensure comparability of evaluation results.

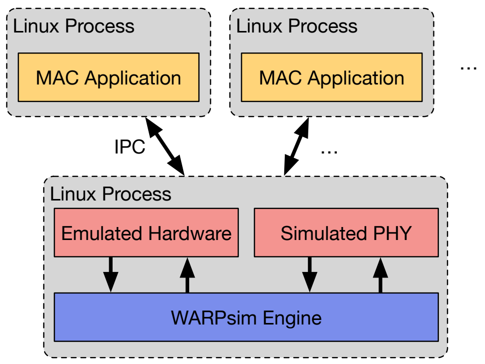
The remainder of this work is structured as follows. In Sec. II, we provide a brief overview of the WARP device that needs to be emulated, as well as the general simulation engine. Sec. III gives an overview of how the audience can interact with the presented demonstration.

II. System Description

The WARP device [7] was developed by the Rice University and Mango Communications to provide an open-



(a) The emulation of the OFDM Reference Design.



(b) The simulation environment of WARPsim.

Fig. 1: The architecture of WARPsim. Most parts of the OFDM Reference Model are emulated by WARPsim (Fig. 1a). Each MAC application instance runs in its own forked process, using IPC for communication to the WARPsim engine (Fig. 1b).

access platform that allows students and researchers to conduct research in wireless communication and especially on the physical and MAC layers. Since the initiation of the project in 2006, three hardware versions of the WARP devices have been released. The main components of a WARP device include:

- a Xilinx Field-Programmable Gate Array (FPGA);
- a microprocessor;
- several clocking resources;
- digital I/O ports, e. g., Ethernet, UART and I/O pins; and
- user I/O, e. g., LEDs, buttons and switches.

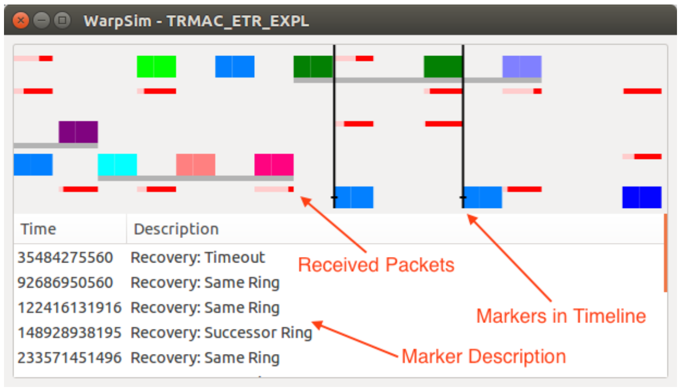
Additionally, the WARP project provides several open-source reference designs that include an FPGA implementation and some basic MAC protocols created using the C programming language. For simulation, we propose WARPsim,[1] a code-transparent network simulator, that emulates a selected subset of hardware used in WARP's OFDM Reference Design to allow the efficient development and implementation of MAC research applications.

A high-level overview of the WARPsim architecture is given in Fig. 1. The simulation engine is a first prototype following a real-time simulation paradigm. As stated in Sec. I, WARPsim is a transparent debug and evaluation tool for MAC protocol researchers. Hence, the MAC application code, which can interchangeably run on actual WARP devices and in the simulator, should not be modified in order to be executed in the different environments. As the simulation exclusively runs on a host Linux machine, the behavior of most reference design components has to be mimicked to realize a coherent behavior. The biggest building blocks are the main simulation engine, a simulated channel, and an emulation of major parts of the OFDM Reference Design. WARPsim is completely written in C, which allows a MAC application to transparently access the same functions of the WARPPHY and WARPMAC API as on the real hardware. However, note that only a selected subset of the API is emulated. The emulation of custom hardware functionality needs to be added if required. WARPsim allows simulating MAC protocols designed for all WARP hardware versions as long as WARP's OFDM Reference Design is employed.

To support evaluation setups with multiple WARP devices, the WARPsim engine can start an arbitrary number of device instances, where each instance runs its MAC protocol in a forked process. Using the shared memory simulation paradigm for the device instances is not possible since the WARP-compatible code allows to create global variables which would be shared by all simulated devices. Hence, the device instances use Interprocess Communication (IPC) to exchange messages with the simulation engine, which likewise runs in its own process. Furthermore, the simulation engine may also communicate to the simulated device instances using IPC, e. g., to forward signals from the simulated channel or the emulated hardware via the WARPPHY and WARPMAC libraries to a MAC application instance. Within each forked process, the typical polling structure of the MAC application is preserved. Callback functions have to be registered as in the OFDM Reference Design, but they are triggered by IPC messages.

The Physical Layer (PHY) of the WARP devices is also part of the emulation in WARPsim. Depending on the MAC protocol, stations send packets of various types at certain points in time. The simulation engine receives the packets via an IPC call from the originating MAC process. In turn, the simulation engine forwards this call to a dedicated process that models the wireless channel on a packet basis. Hence, it entirely abstracts from OFDM specifics and allows to study different PHYs as transmission lengths of packets can be adjusted. Symbol level or even waveform level simulation may be added if required. The channel simulation process keeps track of the current state of the channel, i. e., if the channel is *free* or *occupied*. In case it is occupied, the channel process indicates which part of a transmission is currently going on, i. e., *preamble*, *packet header* and *packet payload*. This is useful in the event of transmission collisions, as it allows to model which parts of the transmission will be affected by the collision. Additionally, depending on the type of wireless channel, the process is able to drop or even corrupt (parts of) packets with a predefined probability p to model transmission errors caused by fading

[1] Available at http://www.comsys.rwth-aachen.de/research/projects/warpsim



Time	Description
35484275560	Recovery: Timeout
92686950560	Recovery: Same Ring
122416131916	Recovery: Same Ring
148928938195	Recovery: Successor Ring
233571451496	Recovery: Same Ring

Fig. 2: A screen shot exemplifying the GUI features, i. e., showing received packets and certain markers for a running MAC protocol.

or interference on the channel. In accordance with the actually simulated condition, the channel process notifies the simulation engine to send the respective messages to the actual device processes via IPC calls such that the corresponding callback functions will be called. In case a packet is lost entirely, no IPC message will be issued for the affected stations.

To provide developers of MAC protocols with enhanced debug and code analysis capabilities, a Graphical User Interface (GUI) is attached to the main simulation engine. During a simulation run, certain IPC messages will be displayed in the GUI by default, e. g., the duration of a transmitted packet or a packet collision. In addition, WARPsim further supports the definition of markers that can be placed at arbitrary positions in the MAC application code. Markers can be thought of as breakpoints in the execution of the protocol, which allow developers to thoroughly analyze the behavior of their MAC application and eventually to detect conceptual errors. A marker consists of an `event type`, an `ID` of the station that invoked the marker and a `timestamp` that stores when the marker was reached (in simulated time). Including these markers can be made code-transparent by adding appropriate preprocessor switches.

To summarize, the simulation engine is responsible for the following tasks:

- handle the IPC among all components;
- start and terminate MAC application instances;
- emulate FPGA and hardware components, as well as selected low-level software functionalities;
- simulate a wireless communication channel;
- provide extended debug capabilities by using code markers; and
- enable aggregation of evaluation metrics.

III. Demonstration Description

In the demonstration, we will show how MAC applications can be simulated using WARPsim. Therefore, we will provide a set of different MAC applications, whose code can be ported readily also to a WARP device. To show that the same application code may be executed on WARP devices and in a simulator, we will use a remote desktop connection to execute it on real WARP devices located in our research lab in Aachen, Germany. Further, we load that same code into the simulator to allow for a deeper inspection of our simulation design. In particular, the audience will be able to select a MAC application and to modify simulation parameters, such as

- the number of stations;
- the emulated latency of WARP's software execution; and
- the channel error probability.

This will enable the audience to observe the impacts of parameter changes on the system behavior depending on the chosen MAC application. This applies in particular to the WARP device-specific parameters. The audience will be able to observe how a false parameterization or how an imprecise emulation of the WARP device will lead to false conclusions about the correctness of a protocol. Moreover, the audience may set code markers at decisive positions in the application code, e. g., once a higher layer signals a packet to be transmitted, to verify the protocol behavior. For this, a GUI window will display for each simulated device a timeline with received packets and illustrate the occurrence of a marker.

In Fig. 2, a screen shot exemplifying the GUI features is shown. The upper part of the window contains the timeline that includes a trace of received packets for each running device instance. When a marker is reached in the code, a colored bar is created at the time of the event. The lower part of the window contains a text field that provides timestamps and precise descriptions given by the programmer. Hence, the audience may scroll through the timeline, zoom in/out and click on markers to highlight their description in order to better understand the behavior of the MAC application.

References

[1] A. Mitschele-Thiel, *Systems Engineering with SDL – Developing Performance-Critical Communication Systems*. John Wiley, 2001.

[2] "Specification and Description Language. ITU-T Z. 100," *International Telecommunications Union, Geneva, Switzerland*, vol. 184, 2000.

[3] "Message Sequence Charts (MSC). ITU-T Z. 120," *International Telecommunications Union, Geneva, Switzerland*, 1996.

[4] K. Wehrle, M. Günes, and J. Gross, *Modeling and Tools for Network Simulation*. Springer Science & Business Media, 2010.

[5] M. Pradella, M. Rossi, and D. Mandrioli, "A UML-Compatible Formal Language for System Architecture Description," in *SDL 2005: Model Driven*. Springer, 2005, pp. 234–246.

[6] R. Ernst, "Codesign of Embedded Systems: Status and Trends," *IEEE Design & Test of Computers*, vol. 15, no. 2, pp. 45–54, 1998.

[7] "WARP Project." [Online]. Available: http://www.warpproject.org

[8] M. Kropff, T. Krop, M. Hollick, P. Mogre, and R. Steinmetz, "A Survey on Real World and Emulation Testbeds for Mobile Ad hoc Nnetworks," in *Int'l Conf. on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM)*, 2006, pp. 447–453.

[9] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 255–270, Dec. 2002.

[10] M. Brumbulli and J. Fischer, "SDL Code Generation for Network Simulators," in *System Analysis and Modeling: About Models*. Springer, 2011, pp. 144–155.