WARP Drive - Accelerating Wireless Multi-hop Cross-layer Experimentation on SDRs

Adrian Loch, Matthias Schulz, and Matthias Hollick

Technische Universität Darmstadt

Secure Mobile Networking Lab

Mornewegstr. 32, 64293 Darmstadt, Germany

{adrian.loch, matthias.schulz, matthias.hollick}@seemoo.tu-darmstadt.de

ABSTRACT

Rapid prototyping of cross-layer multi-hop schemes in wireless networks often poses a hard challenge. While Software-defined Radios (SDRs) allow to implement virtually any cross-layer technique, the underlying programming models for rapid prototyping are inherently designed for one-hop communication. Moreover, existing solutions are typically non-real-time and fall back to offline processing. While this is well-suited for evaluating techniques at the physical layer, the media access control and network layers demand interactivity. Further, network size becomes a critical parameter in multi-hop settings but the size of SDR testbeds is often limited, requiring costly reimplementations in a network simulator to investigate larger settings. We develop a framework to overcome these limitations and enable rapid prototyping of cross-layer multi-hop mechanisms on SDRs. We build on (1) a modular software design to allow for mechanism exchange, (2) a virtual timeline to abstract from the non-real-time nature of transmissions, and (3) a seamless switch from practical experiments to simulations using the same code. We provide a reference implementation of our framework to the community as a starting point for rapid prototyping of cross-layer multi-hop mechanisms.

Categories and Subject Descriptors

C.2.1 [Computer-Communication Networks]: Network Architecture and Design—*Wireless Communication*

Keywords

rapid prototyping; wireless multi-hop networks; cross-layer

1. INTRODUCTION

Rapid prototyping on Software-defined Radios (SDRs) is limited when it comes to cross-layer mechanisms beyond one-hop wireless communication. Although there exist frameworks that facilitate implementation on SDRs, the underlying programming model often relies on offline processing.

This impedes interactive protocol operation. For example, the WARPLab framework for the Wireless Open-Access Research Platform (WARP) [2] SDR allows for flexible implementation in Matlab. Still, it incurs significant delays for transferring signals to the SDR and processes data offline.

Such an offline approach is well suited for evaluating physical layer techniques on real-world channels. However, it poses a number of significant challenges to cross-layer mechanisms spanning up to the network layer. First, Media Access Control (MAC) and network mechanisms often demand interactivity, e.g., answering requests from other nodes. This is not possible when operating offline. Secondly, network size is a critical parameter in multi-hop communication. The size of SDR testbeds is typically limited and thus simulations are often utilized to investigate larger settings. This translates into implementing the system twice, that is, once on the SDR and once for the simulator. Finally, SDR rapid prototyping frameworks are not designed for supporting multiple layers of a protocol stack. Thus, the resulting code is not modular, but needs to be highly bonded to the SDR prototyping framework itself on all layers to support cross-layer interactions. This causes a significant maintenance overhead and hinders extensibility.

Existing frameworks that enable implementation of cross-layer mechanisms are often limited to traditional physical layers such as Orthogonal Frequency-Division Multiplexing (OFDM). For example, this is the case for the WARP 802.11 reference design [4] and similar frameworks for GNU Radio [7]. Support for advanced lower layers such as Orthogonal Frequency-Division Multiple Access (OFDMA), Interference Alignment (IA) or Multiple-Input Multiple-Output (MIMO) as used in 802.11n is not available. For WARP, the alternative to the aforementioned non-real-time approach is realizing a system on its FPGA. While such a prototype is desired, the implementation is very costly in terms of time. A rapid prototyping framework which facilitates the implementation of cross-layer mechanisms from the network layer down to the physical layer for multi-hop networks is missing.

In this paper we present *WARP Drive*, a framework that accelerates prototyping of cross-layer multi-hop mechanisms on SDRs. It builds on WARPLab but overcomes the aforementioned limitations using an event-driven programming model. Each transmission becomes an event and can be processed in non-real-time while still allowing for interactivity. In particular, our framework schedules these events in a *virtual timeline* and processes them sequentially. This also allows for adaptation to, for instance, channel characteristics as long as channels change slowly, such as in a lab environ-

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author. Copyright is held by the owner/author(s). *SRIF'14*, August 18, 2014, Chicago, IL, USA.
ACM 978-1-4503-2995-8/14/08.
<http://dx.doi.org/10.1145/2627788.2627800>.

ment. The execution of an event can generate new events which are added dynamically to the timeline, e.g., answering requests from other nodes. Mechanisms associated to events are encapsulated in well-defined blocks, resulting in modular code. Additionally, *WARP Drive* can send interchangeably via actual WARP boards or simulated channels on the same code basis. This common interface also decouples code from changes and version updates of WARPLab. In summary, our contributions are as follows:

- We propose a modular framework for rapid prototyping of cross-layer multi-hop mechanisms on WARP.
- We enable seamless switching from practical experiments to simulations for a complete network¹.
- We provide an implementation of *WARP Drive*² to the scientific community for experimentation.

This paper is structured as follows. First, we survey existing SDR frameworks in Section 2. Next, we present a general view on the architecture of our framework in Section 3. In Section 4 we provide performance results of our system in terms of processing time on third-generation WARP boards. Finally, in Section 5 we conclude the paper.

2. RELATED WORK AND BACKGROUND

The flexibility of SDRs comes at the price of processing delays, since functionality is realized in software instead of application specific hardware. These delays hinder interactive protocols at the MAC and network layers. However, certain SDR architectures [12, 2] execute code closer to the hardware than others [16, 3], resulting in different delays. Current approaches can coarsely be categorized in two classes, namely FPGA-based and host-based.

First, **FPGA-based** solutions implement most functionality on FPGAs directly on the SDR. This results in very low processing delays at the price of a much higher implementation effort compared to host-based approaches. Examples include the WARP and Airblue SDR platforms. The former achieves real-time performance [4, 5], while the latter focuses on a modular design. Secondly, **host-based** solutions implement core functionality on General-Purpose Processors (GPPs). This approach is highly attractive since it allows for rapid prototyping of solutions—at the cost of a higher processing delay. The SORA [16] SDR aims at minimizing this delay by storing pre-computed waveforms of time-critical data. The widely used GNU Radio [1] with the USRP [3] SDR incurs higher delays than SORA but in turn features easier extensibility. While GNU Radio requires implementation in C++, WARPLab [6] takes rapid prototyping a step further allowing for implementation in Matlab. This is highly convenient for prototyping complex signal processing schemes but also places WARPLab at the highest end of the scale for processing delays.

SDR processing delays are extensively studied in [15, 14] for the case of GNU Radio with USRP. Both analyze the feasibility of 802.11 and conclude that the minimal receive latency is already larger than the inter-frame spacing requirements of current MAC protocols. This issue becomes

¹The WARP project [2] provides a similar functionality but is limited to two nodes instead of an arbitrary number.

²Available at <http://www.seemoo.de/warpdrive>

particularly critical for sophisticated MAC layers such as [17] which are based on, e.g., machine-learning techniques for cognitive radio. The authors of [13] categorize MAC functionality based on its timing constraints. Essentially, they propose to split MAC protocols and implement critical components close to the hardware while leaving the rest to GPPs. Other approaches such as CalRadio [9] also deal with the MAC layer but assume a fixed 802.11 physical layer, which limits their flexibility compared to SDRs.

While the aforementioned research efforts focus on the challenges resulting from SDR processing delays at the MAC layer, these issues also propagate to the network layer. Existing SDR *network* testbeds [11] use FPGA-based solutions to achieve real-time performance, which raises the bar regarding implementation effort. The alternative proposed in [11] is resorting to standard 802.11 hardware, thus limiting cross-layer implementations to the MAC and network layers. The authors of [8] present a framework which brings together the USRP SDR with the Click Modular Router [10], i.e., they provide a tool that spans up to the network layer. However, [8] focuses on the MAC layer and is still limited by the processing delay of the USRP. While the connection to Click allows for network layer functionality, the design of [8] is tailored to one-hop wireless communication. A framework for rapid prototyping of cross-layer multi-hop protocols on SDRs that solves the impact of long processing times is missing. We propose an approach to close this gap.

3. ARCHITECTURE

The architecture of *WARP Drive* comprises two components, namely the modular network stack and the WARPLab interface. The latter builds on the WARPLab framework provided by the WARP Project [2], which in turn interacts with the WARP hardware itself. Figure 1 gives an overview on all components of our architecture.

3.1 WARPLab

The WARPLab framework allows to interact with the WARP boards via Matlab. Essentially, a central computer processes the down-converted raw samples of all boards in the testbed. For each transmission, the computer calculates

Figure 1: *WARP Drive* architecture.

the transmit samples and sends them via Ethernet to the transmitting board. This board transmits the samples over the actual wireless medium to the receiver board, which in turn sends the received signal again via Ethernet to the central computer for further processing.

3.2 WARPLab interface

Our interface for WARPLab follows an object-oriented approach to model the underlying hardware. It provides a fixed set of functions to the code running on top of it, which thus does not need to be changed whenever the underlying WARPLab is updated. The current implementation of our interface supports WARPLab versions 6 and 7.

3.2.1 Node abstraction

The interface abstracts from the underlying hardware. In particular, it provides node entities with one or multiple antennas that can be mapped to any combination of WARP antennas in the testbed. In particular, we can map antennas according to three patterns, as depicted in Figure 2.

- **Antenna Combining.** All antennas of two or more boards are combined to one multi-antenna node. This allows, for instance, large MIMO systems.
- **Antenna Splitting.** A subset of the antennas of one board is defined as one node. This allows for example defining each antenna of a board as an individual node to obtain larger networks with fewer boards.
- **Antenna Picking.** Subsets of antennas of multiple boards are defined as one node, resulting in a combination of antenna combining and splitting. This allows for high flexibility, as any configuration is possible.

Combining antennas of different nodes raises practical issues regarding synchronization of the underlying boards. However, the WARP Project provides means for both wired time and frequency synchronization.

Figure 2: Node abstractions. The shaded areas represent node entities formed by multiple boards.

3.2.2 Modular design

Our interface modularizes hardware elements such as radios, boards and the complete testbed to make code changes simple and efficient. The aforementioned node abstraction builds on these modules, allowing to change antenna mappings easily. Figure 3 shows how the modules compose our interface. *CWarplab* comprises all WARP boards involved in the experiment, triggers the transmission of samples, and handles channel simulation when running experiments offline. Next, *CWarpnode* represents a WARP board and all its parameters, such as channel selection or Automatic Gain Control (AGC). Moreover, it sets up and transfers samples to the board. Each board incorporates one *CRadio* module per antenna. This module controls the baseband and radio frequency amplifier gains of each radio. Finally, the *CNode* module maps WARP antennas to abstract nodes. It spans multiple *CWarpnode* modules, as it may include antennas from any testbed board. Our reference implementation of the interface focuses on OFDM-based wireless systems and thus includes basic OFDM functionality in *CNode*, such as preamble detection, Carrier-Frequency Offset (CFO) correction, and Fast Fourier Transform (FFT). However, the architecture of our WARPLab interface is valid also for other approaches.

3.2.3 Switching to simulations

As shown in Figure 1, our interface can transmit samples both over simulated channels or the actual wireless medium. Switching between both modes is seamless and does not require any code change. Simulations are not limited to one transmitter and one receiver, but are possible for any network size N . In particular, we model all $N \cdot (N - 1) / 2$ links of the network as Rayleigh channels but other channel models can be incorporated. Each link is associated to an average Signal-to-Noise Ratio (SNR) value which determines its quality. That is, our model is based on a fully-connected network to account for any possible interaction between nodes at the physical layer. We assign low SNRs to far away node pairs, or otherwise impaired links, which protocols in our modular network stack identify as non-existing links.

We represent the simulated channels of our fully-connected network model as a five-dimensional array. The first dimension stores the channel impulse response, while the remaining four dimensions account for the transmit board, the transmit antenna, the receive board, and the receive antenna that form the link the impulse response refers to. That is, the array reflects exactly the same links between the WARP boards that would compose the testbed in the practical case. The received signal at each antenna is computed as the sum

Figure 3: WARPLab interface modules for a testbed with two WARP boards and three node entities.

in the time domain of all sent signals, convolved with their respective channels. As a result, our simulation considers the interference caused by any transmitter on any receiver, both intended (e.g., MIMO or Interference Alignment) or accidental (e.g., interfering parallel transmissions in a wireless multi-hop network).

3.3 Modular network stack

In the following, we describe the architecture of our modular network stack. We designed it in Matlab, based on the node abstraction provided by the WARPLab interface.

3.3.1 Assumptions

Similar to most host-based solutions, WARPLab incurs significant delays for transferring samples to the aforementioned central computer and for processing data in Matlab. We cannot solve this inherent limitation but we circumvent it assuming a static scenario, i.e., we consider that network changes occur at a larger timescale than the delays. While this would be a strong assumption for production systems, it is realistic for a lab environment and thus suitable for our rapid prototyping framework. Essentially, we process data and run protocols similarly to a production system but much slower. In turn we can implement protocols on all layers in Matlab, which speeds up prototyping and allows to get quick insights into performance of wireless multi-hop networks.

3.3.2 Virtual timeline

To allow for interactive MAC and network layer protocols despite the aforementioned processing delays, we use a *virtual timeline*. Intuitively, the virtual timeline keeps track of how long transmissions would take without the processing delays incurred by WARPLab. The resulting timing values can be used for, e.g., estimating throughput. In particular, the timeline accounts for the data transmission time, that is, for the time the medium is in use. To this end, we define events and actions as follows.

- **Events.** We call data transmissions that occupy the medium *events*. Events can be scheduled on the timeline to happen at any point in time, as long as the medium is not yet occupied by a different transmission at that time.
- **Actions.** In contrast, we call the processing at nodes *actions*. The timeline does not account for the processing time of actions, since this is specific to the implementation. For instance, in most cases a prototype implementation of an algorithm in Matlab has a longer processing time than a production version on optimized hardware.

Events are related to the physical layer, while actions refer to the rest of the stack. As events are processed, the *testbed time* advances. Both actions and events can trigger new events which are scheduled in the timeline as soon as possible, either after the current testbed time or after a given point in time. The latter allows to schedule transmissions in the future. In both cases, the timeline finds a suitable gap for the event. For instance, in Figure 4(a) Event 3 does not fit in between Event 1 and Event 2, and is thus scheduled after Event 2. The duration of events is estimated during scheduling but can be corrected later. Explicit synchronization of actions and events is not needed since both include all parameters needed to be executed individually.

3.3.3 Processing loop

Figure 4(b) depicts the main processing loop of our modular protocol stack. Each loop iteration processes one event, that is, one transmission takes place and the testbed time advances to the point in time immediately following it. Additionally, all pending actions are executed. If there are no more pending events or actions, the experiment finishes. Both actions and events are code modules that take a certain set of parameters as inputs and typically generate new actions/events that continue processing their output. The boundaries of each module are flexible, that is, cross-layer interactions ranging from soft information to fully merged layers are possible throughout the protocol stack. As long as modules are compatible to each other, they also can be exchanged flexibly. For instance, this allows to prototype wireless multi-hop networks that switch among protocols on one or multiple layers of the stack.

4. BENCHMARKS

We provide a reference implementation of *WARP Drive*, which is available to the community as a basis to build wireless multi-hop prototypes on top of it. In this section, we evaluate its performance in terms of processing delays. This is crucial since we assume the coherence time to be longer than the delay. In other words, this section gives an intuition on how stable channels in a lab environment must be in order to guarantee that our framework operates correctly.

4.1 Reference implementation

Our reference implementation includes essential functionality at the physical (PHY), MAC, network (NET), and upper layers (UPP). We divide the physical layer in “lower” and “higher”—the former transmits IQ-modulated data symbols using OFDM while the latter handles modulation and coding of binary data at a fixed rate. Moreover, the higher physical layer enables unicast, multicast, and broadcast addressing, as well as frame fragmentation. The reference MAC layer triggers retransmissions and provides adaptive modulation and coding. Finally, the reference network layer forwards data assuming routing state is available.

4.2 Results

Time-critical functions typically reside at the physical and MAC layers since the network layer operates at a much larger time scale. Thus, we quantize the processing delay in a single-hop scenario with one transmitter and one receiver. In particular, we measure the transmission delay at each layer for both simulation and practical transmissions using WARP hardware. Figure 5 shows our results.

Figure 4: Event/action processing and scheduling.

Figure 5: Processing delay at each layer for one hop.

As expected, we observe that the delay in simulation is consistently smaller than for the practical case. The reason is that no data is sent to the WARP boards and thus no delay for transferring samples is incurred. However, at higher layers the difference becomes smaller as simulations take more time—simulations use randomly generated channels and thus require retransmissions more often, whereas our practical testbed provides good channel conditions most of the times. More interestingly, Figure 5 shows a significant difference in delay between lower and higher physical layers. The underlying cause is the convolutional encoder, which in our implementation is part of the higher physical layer. Thus, the overall delay of about 0.4 s is to a large extent due to Matlab. The delay for transferring samples to the WARP boards is an order of magnitude smaller, at only 35.1 ms.

We conclude that processing in non-real-time using *WARP Drive* is feasible as long as the coherence time of the channel is at least about 70 ms, which is the minimum delay incurred at the lowest layer. While further processing at higher layers requires larger coherence times, this is often feasible in lab environments. Still, it is crucial to assess the delay of each particular *WARP Drive* system to determine if it is smaller than the coherence time, and thus measurements are valid.

5. CONCLUSIONS

We present *WARP Drive*, a framework for rapid prototyping of cross-layer protocols for wireless multi-hop networks on the WARP SDR platform. We build on WARPLab, which is a reference design for WARP that enables flexible design of physical layer prototypes in Matlab. It processes data offline and thus does not allow for the interactivity required by MAC, network, and upper layers. *WARP Drive* avoids this limitation by using a “virtual timeline” to model the behavior of a real-time implementation using event-driven programming.

Additionally, we present a modular network stack that enables protocols to span multiple layers, a modular interface for WARPLab that allows high flexibility regarding testbed configuration, and a seamless switch from practical to simulation results on the same code basis. We provide an implementation of *WARP Drive* to the community for experimentation, including reference code for a wireless multi-hop network as a starting point for researchers using *WARP Drive*.

Acknowledgments

This work was supported by the LOEWE Priority Program Cocoon (www.cocoon.tu-darmstadt.de), the DFG CRC 1053 MAKI (www.maki.tu-darmstadt.de) and LOEWE CASED (www.cased.de).

6. REFERENCES

- [1] GNU Radio. <http://www.gnu.org/software/gnuradio/>.
- [2] Rice University WARP Project. <http://warp.rice.edu>.
- [3] USRP. The universal software radio peripheral. <http://www.ettus.com/>.
- [4] WARP 802.11 Reference Design. <http://warpproject.org/trac/wiki/802.11>.
- [5] WARP OFDM Reference Design. <http://warpproject.org/trac/wiki/OFDMReferenceDesign>.
- [6] WARPLab Reference Design. <http://warpproject.org/trac/wiki/WARPLab>.
- [7] B. Bloessl, M. Segata, C. Sommer, and F. Dressler. An IEEE 802.11a/g/p OFDM Receiver for GNU Radio. In *Proc. of the Second Workshop on Software Radio Implementation Forum*, 2013.
- [8] R. Dhar, G. George, A. Malani, and P. Steenkiste. Supporting Integrated MAC and PHY Software Development for the USRP SDR. In *1st IEEE Workshop on Networking Technologies for Software Defined Radio Networks*, 2006.
- [9] A. Jow, C. Schurgers, and D. Palmer. CalRadio: A Portable, Flexible 802.11 Wireless Research Platform. In *Proc. of the 1st International Workshop on System Evaluation for Mobile Platforms*, 2007.
- [10] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. *ACM Trans. Comput. Syst.*, 18(3), 2000.
- [11] T. Korakis, M. Knox, E. Erkip, and S. Panwar. Cooperative Network Implementation using Open-Source Platforms. *Comm. Magazine*, 47(2), 2009.
- [12] M. C. Ng, K. E. Fleming, M. Vutukuru, S. Gross, Arvind, and H. Balakrishnan. Airblue: A System for Cross-layer Wireless Protocol Development. In *Proc. of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2010.
- [13] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste. Enabling MAC Protocol Implementations on Software-defined Radios. In *Proc. of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2009.
- [14] A. Puschmann, M. A. Kalil, and A. Mitschele-Thiel. Implementation and Evaluation of a Practical SDR Testbed. In *Proc. of the 4th Int. Conf. on Cognitive Radio and Advanced Spectrum Management*, 2011.
- [15] T. Schmid, O. Sekkat, and M. B. Srivastava. An Experimental Study of Network Performance Impact of Increased Latency in Software Defined Radios. In *Proc. of the Second ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*, 2007.
- [16] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang, and G. M. Voelker. Sora: High Performance Software Radio Using General Purpose Multi-core Processors. In *Proc. of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 2010.
- [17] X. Zhang, J. Ansari, M. Arya, and P. Mähönen. Exploring Parallelization for Medium Access Schemes on Many-core Software Defined Radio Architecture. In *Proc. of the Second Workshop on Software Radio Implementation Forum*, 2013.