ViPMesh: A Virtual Prototyping Framework for IEEE 802.11s Wireless Mesh Networks

Michael Rethfeldt, Hannes Raddatz, Benjamin Beichler,
Björn Konieczek, Dirk Timmermann, Christian Haubelt
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany, Tel.: +49 381 498-7269
Email: michael.rethfeldt@uni-rostock.de

Peter Danielis
ACCESS Linnaeus Center, School of Electrical Engineering
KTH Royal Institute of Technology, Stockholm, Sweden
Email: pdanieli@kth.se

***Abstract*—WLAN mesh networks are characterized by their flexible and low-cost deployment, scalability, and self-healing capabilities. The new WLAN standard IEEE 802.11s introduces low-level mesh interoperability. However, building large-scale real-world test beds and reproducible setups is challenging and costly. In the majority of research works, network simulation is preferred over practical measurements. Here, the main disadvantage exists in simplified device and protocol models restricting the comparability to practical implementations. In contrast, using device emulation still requires the simulation of wireless channel and environment models. Consequently, a combination of both emulation and simulation is needed to enable virtual prototyping of real applications and protocols in WLAN mesh networks. Nevertheless, the computation of complex wireless channel effects requires a decoupling of wall clock and simulation time. Therefore, we present *ViPMesh*, a virtual prototyping framework for IEEE 802.11s and its Linux reference implementation. ViPMesh relies on WLAN device emulation and nested virtualization using QEMU and Linux containers to support the analysis of real applications on top of an unmodified protocol stack. Adopting an alternative time source approach for QEMU, ViPMesh acts as discrete-event simulator. It further integrates channel and environment models with support for IEEE 802.11n MIMO techniques, high throughput modes, multi-channel operation, and node mobility. To the best of our knowledge, this is the first approach that combines the IEEE 802.11s reference implementation with the described simulation features. The functionality of ViPMesh is demonstrated in different example scenarios.**

I. Introduction

Compared to common centralized WLAN infrastructures, WLAN mesh networks feature automatic peering and multi-hop routing and thus provide a flexible and low-cost wireless network extension with high scalability and robustness. Since 2011, the new standard amendment IEEE 802.11s enables low-level interoperability, integrating mesh mechanisms directly into the WLAN MAC layer [1]. As a promising core technology for future wireless communication networks, IEEE 802.11s is subject to ongoing research that aims at optimizing the interplay with existing network protocols and applications. Thereby, we pursue the evaluation of network applications as well as the prototyping of own optimization algorithms, developed specifically for IEEE 802.11s networks. However, the setup of real-world test beds is costly, impracticable, or simply not possible for increasing network sizes and dynamics. Network emulation or simulation allow for reproducible measurements also for flexible setups with a large number of nodes [2]. Above all, emulation permits software design and analysis on top of an unmodified protocol stack, to leverage the comparability with real-world test beds. However, it still needs to be combined with simulations that account for wireless channel effects, different propagation environments, or dynamic network topologies due to node mobility. Furthermore, the integration of comprehensive simulation models is desirable to investigate the influence of modern WLAN technology parameters, such as IEEE 802.11n MIMO techniques, or high throughput (HT) configurations. As a result, the considerable computational effort of complex simulation models leads to the requirement of synchronizing both simulation and system time of emulated network participants. We denote the combination of node emulation and network simulation as a *virtual prototyping* approach for the evaluation of applications and algorithms for IEEE 802.11s WLAN mesh networks.

Consequently, we present *ViPMesh*, a virtual prototyping framework for IEEE 802.11s and its Linux reference implementation. ViPMesh relies on WLAN device emulation and nested virtualization using QEMU and Linux containers to support the analysis of real applications on top of an unmodified protocol stack. Adopting an alternative time source approach for QEMU, ViPMesh acts as discrete-event simulator. As major contribution, it complements the IEEE 802.11s reference implementation with comprehensive models for medium access, physical-layer/environment effects, multi-channel operation, and mobility.

The objectives of our approach are as follows:

- Virtual prototyping of IEEE 802.11s mesh networks
- Evaluation of real applications and protocol stacks
- Realistic (multi-)channel and mobility simulation
- Modelling of modern physical-layer features
- Independence of simulation performance and emulation

The remainder of this paper is organized as follows: Section II introduces the technological basis. Section III formulates the problem statement. In Section VI, we provide an overview of previous work in the field of combined simulation/emulation, and reason why current solutions are not sufficient to fulfill the objectives mentioned above. Section IV presents the proposed ViPMesh framework followed by Section V, which shows results and discusses them. Finally, we conclude the study in Section VII, and suggest directions for future work.

II. Technological basis

In this Section, we first provide the fundamentals of IEEE 802.11s WLAN mesh networks and the Linux reference implementation of the standard. Second, we give an overview of approaches for network protocol and application analysis.

A. IEEE 802.11s WLAN Mesh Networks

The mesh standard IEEE 802.11s comes as amendment to the 802.11 MAC-layer specification and thus inherits most of its mechanisms. It enables vendor-independent infrastructure-less multi-hop communication based on the widespread WLAN technology. For this, it introduces mesh extensions, such as automatic link establishment (peering), frame forwarding, and path selection (routing) [1]. To ensure interoperability, every node must support the *Hybrid Wireless Mesh Protocol* (HWMP) and the *Airtime Link Metric* (ALM) as default routing mechanisms. Thereby, ALM represents a cost metric for transmitting a frame over a specific mesh link by considering the applied WLAN physical layer and the wireless medium. The Linux project *open80211s* [3] is currently the most advanced open-source reference implementation of IEEE 802.11s. It is part of the *mac80211* kernel module, representing the Linux software WLAN MAC layer.

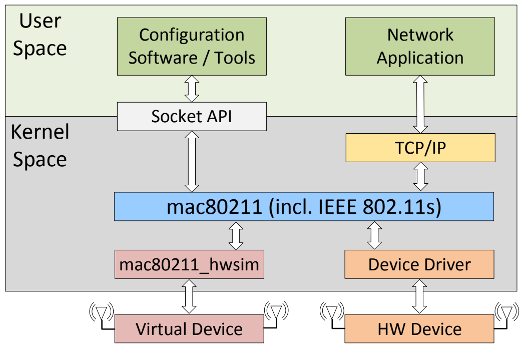

Fig. 1: Linux kernel with mac80211 and mac80211_hwsim

Figure 1 depicts *mac80211* within the Linux kernel. Apart from the integration with the TCP/IP protocol stack, a socket API provides direct access for userspace software, e.g., to configure device or MAC-layer parameters. *mac80211* is used either by real WLAN hardware, if compatible drivers exist, or by emulated virtual WLAN devices, created with the help of kernel module *mac80211_hwsim*. As designed only for the functional testing of *mac80211*, frames generated by *mac80211_hwsim* devices are assumed to be sent over a perfect channel, i.e., transmissions are always successful and issued immediately. Thus, the open-source software *wmediumd* [4] was developed to complement *mac80211_hwsim* with a wireless medium simulator, running in userspace. In its initial version, *wmediumd* only applies a configurable loss probability to frame transmissions. In 2013, an extension was released that enables step-wise mobility calculation and adds a simplified path loss model with static interference [5].

Consequently, the combination of *mac80211_hwsim* and *wmediumd* represents a first promising step towards a prototyping framework for IEEE 802.11s networks. However, in its original version, *wmediumd* only provides a rudimentary channel model and does not account for a simulation time that is independent of system performance. Following our objectives, stated in Section I, a new approach for integrating and extending *wmediumd* is necessary.

B. Approaches for Evaluating Practical Network Protocols and Applications

In contrast to the pure simulation of network protocol and application models within simplified scenarios, different options for the evaluation of real implementations exist. In [2] a classification is given, distinguishing the following approaches:

- One class of approaches uses real or emulated systems as data source for network and protocol simulations. However, the system behavior depends on the simulation performance and there is no decoupling of wall clock and simulation time, which may lead to considerable inaccuracies. Thereby, *wall clock* denotes the real-world time of the simulation host, whereas *simulation time* denotes the virtual time considered in the network model.
- Another possibility is the integration of protocol implementations, extracted from real systems, into simulations. Here, the result is a high maintenance effort to keep track of implementation updates and perform adaptations for the respective simulation environment. Moreover, evaluation is still less accurate compared to the integration of full real-world systems and protocol stacks.
- The most promising approach is found in the integration of virtualized systems into simulations, combined with a decoupling of wall clock and simulation time. This provides for the highest accuracy, as the simulation models only need to cover hardware- and physical-level effects that are not part of the emulation. Moreover, maintenance effort is kept low, since implementation updates can be deployed by simply replacing virtualized systems, system parts, or programs. As a result, the *virtual prototyping* of real applications on top of unmodified protocol stacks is made possible, as pursued in our work.

For the third class, different levels of virtualization can be considered. In the course of this paper, we distinguish *system virtualization* and *container virtualization*. Whereas the first denotes the virtualization of a complete operating system (OS) within an emulated machine, the latter represents a more lightweight variant by isolating only system components such as file systems, processes, or network stacks on top of a shared OS kernel. Practical implementation examples are, e.g., the open-source emulator QEMU for system virtualization [6], and Linux Containers (LXC) for container virtualization [7]. In our architectural concept, presented in Section IV-A, we combine both virtualization methods.

III. Problem Statement

In Section I, we outlined the main objectives of our approach. To fulfill these objectives, a joint simulation/emulation environment should meet the following functional requirements:

- Support of IEEE 802.11s
- Support of IEEE 802.11n MIMO and HT
- Support of real applications
- Support of a real protocol stack
- Support of wireless channel effects
- Support of environmental effects
- Support of multi-channel operation
- Support of device mobility

Table I compares a selection of widely used discrete-event simulators with respect to the given requirements. These simulators have been selected as they either are open source or can be freely used for academic applications. Features marked as "partly met" have to be considered as model properties with insufficient coverage of the objective. For example, *ns-3* only supports co-channel interference and therefore does not permit the realistic evaluation of multi-channel setups, whereas *OMNeT++* only provides an incomplete IEEE 802.11s model. The comparison shows that currently there is no simulator, which meets all the requirements mentioned above.

The simulator *wmediumd* models a lossy wireless medium between virtualized WLAN devices under Linux [4]; its mobility extension allows for simulating step-wise node mobility [5]. *wmediumd* was developed to complement the WLAN device emulator *mac80211_hwsim* which is part of the Linux kernel (see Section II-A). It utilizes the real-world Linux WLAN stack along with its IEEE 802.11s reference implementation. Thus, *wmediumd* has been chosen as basis for our framework.

IV. ViPMesh Framework

In this Section we present our virtual prototyping framework *ViPMesh* for IEEE 802.11s WLAN mesh network applications. We pursue a joint emulation/simulation approach with nested virtualization, as given by the classification in Section II-B. First, we outline our architectural concept, distinguish emulated and simulated components, and explain levels of virtualization. Second, we illustrate the decoupling of wall clock and simulation time, originating from [2]. Third, we describe implementation details of our simulation models.

A. Architectural Concept

The architecture of *ViPMesh* is shown in Figure 2. It is based on multiple virtualization levels and mainly comprises:

- the host OS, running a native simulation daemon and the guest OS, running inside a VM (full virtualization)
- the guest VM, containing all emulated mesh nodes within separate network namespaces (containers)
- interfaces for host-guest information exchange
- a control protocol for the decoupling of wall clock and simulation time, according to [2]

To virtualize the guest OS, building the emulation part of our framework, we rely on the open-source software QEMU [6]. This first virtualization step is needed to decouple wall clock and simulation time, as detailed in Section IV-B. Inside the guest VM, we run a Linux OS that emulates arbitrary numbers of mesh nodes with real protocol stacks and applications. The WLAN MAC layer is located within the Linux kernel module

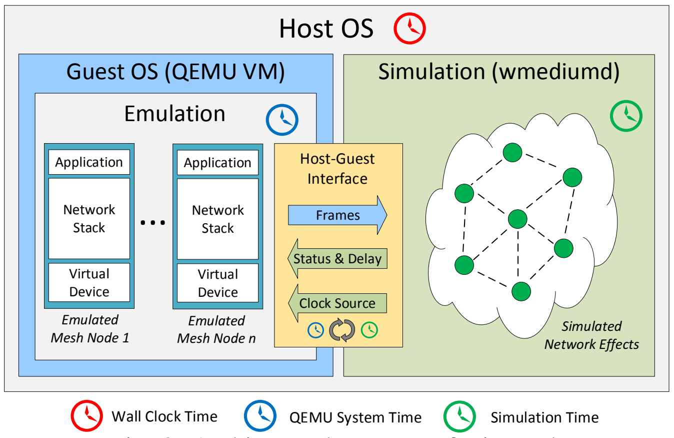

Fig. 2: Architectural concept of ViPMesh

mac80211 that also integrates the IEEE 802.11s reference implementation. A second kernel module, *mac80211_hwsim*, allows for the creation of virtual WLAN devices, as introduced in Section II-A. Thus, despite our particular focus on IEEE 802.11s, this approach also allows for the evaluation of common infrastructure-mode WLAN setups or any other MAC-layer variant, as configurable under Linux. Using the concept of Linux containers (LXC), we assign each virtual device to a separate network namespace and thereby obtain isolated protocol stacks and applications [7]. In contrast to approaches that put each emulated node in a separate VM, container virtualization represents a lightweight alternative and allows for good scalability when emulating large network setups (see Section II-B). However, if heterogeneous systems shall be evaluated, a new VM per hardware architecture would be required, that still emulates all nodes of the same platform using container virtualization.

A simulation daemon within the host OS is responsible for processing network topology, mobility, environment/channel effects, and medium access delays. All emulated mesh nodes in the guest VM are mapped to the corresponding simulated node instances on the host side. Communication channels are used to exchange information about frames that are generated by the emulated nodes (guest to host) as well as transmission status and duration, determined by the simulation (host to guest). The host-guest interfaces are based on socket communication, using the *VirtIO* framework [13]. An additional channel is used for time synchronization between simulation and QEMU instance. Our simulation daemon originates from the open-source software *wmediumd* [4], as introduced in Section II-A. Adopting its mobility support, added by an external project [5], we heavily extended and replaced the *wmediumd* internals with our own simulation models for physical-layer effects, IEEE 802.11n MIMO techniques, and medium access (IEEE 802.11e EDCA). Further details are given in Section IV-D.

B. Decoupling of Wall Clock and Simulation Time

To prevent the computing performance of the host system from affecting the combined emulation/simulation process, a decoupling of wall clock and simulation time is necessary.

TABLE I: Comparison of selected discrete-event simulators with respect to our requirements (+: req. is met, o: req. is partly met, -: req. is not met)

Requirement	NS-3 [8]	GloMoSim [9]	OMNeT++ [10]	NCTUns [11]	SWANS [12]	wmediumd [4]	wmediumd mob. ext. [5]
IEEE 802.11s	+	-	o	-	-	+	+
Real applications	+	-	-	+	o	+	+
Real protocol stack	-	-	-	+	-	+	+
MIMO techniques	-	-	-	-	-	-	-
Interference effects	o	+	+	+	+	-	o
Multi-channel operation	+	-	o	+	-	-	-
Device mobility	+	+	+	+	+	-	+

We adopt an approach originating from Werthmann et al. [2], comprising the control protocol *IKR SimLib* along with a corresponding patch for the QEMU emulator. Providing an alternative clock source for QEMU, the guest VM's real-time clock (RTC) and high precision event timer (HPET) are advanced in discrete time steps, controlled by the simulation in the host OS. In between time steps, the guest VM handles external I/O communication and executes processes until they are blocked by I/O requests, alarms, or timer events. Consequently, the guest VM automatically applies the simulation time as its system time and ViPMesh acts as discrete-event simulator. As a side effect, the influence of computation and processing time inside the guest OS gets lost, while the impact of network effects is emphasized. However, our focus is on analyzing the behavior of real applications and protocols depending on the underlying communication technology, channel effects, and network topology. Moreover, we assume these network effects to dominate the overall communication latency.

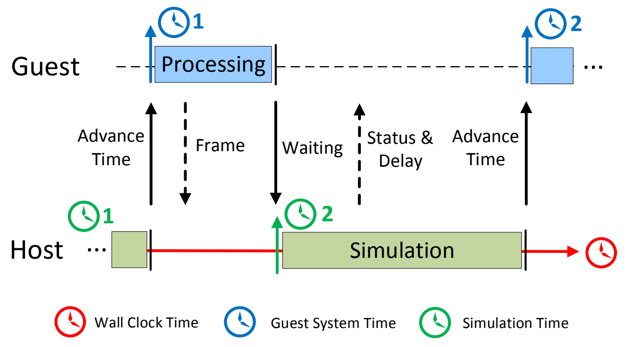

Fig. 3: Emulation/Simulation Procedure of ViPMesh

C. Emulation/Simulation Procedure

In Figure 3, the basic emulation/simulation procedure for a discrete advance of simulation time including a frame transmission is depicted, showing the relation between the simulation time of the host OS and the system time of the guest VM. We illustrate the discrete advance of simulation time (green) and guest system time (blue) as arrows whereas the host OS wall clock time (red) is displayed as continuous timeline. Additionally, the processing times needed by emulation and simulation, as related to the wall clock time without influence on simulation and guest time, are expressed as bars.

A frame generated by an emulated node of the guest VM is passed to the host OS, along with further transmission information required, such as the current data rate or WLAN channel. On the host side, the annotated frame travels through several simulation steps. Results of the simulation are an information message, denoting transmission status (success or failure) to the frame originator and a message containing the actual frame for the destination node. Moreover, overall frame delay for transmitter and receiver is included in the messages and reported back to the emulation. The guest VM then applies the actual delivery of all successful frames to the emulated nodes, as determined by the simulation.

D. Simulation Models

In the following, we briefly describe the different simulation models that were integrated into *wmediumd*. Figure 4 shows the order of models, as applied to each frame generated by the emulation. Thereby, each model adds a virtual delay and/or attenuation of the signal-to-noise ratio (SNR) to the simulated frame transmission.

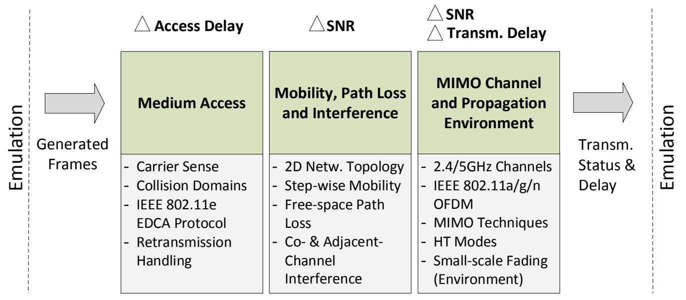

Fig. 4: Simulation models of ViPMesh

Starting with a *Medium Access* model to derive the random back-off delay caused by contention-based channel access, SNR attenuation effects depending on spatial and spectral distance are determined by the *Mobility, Path Loss and Interference* model. As a last step, technology-specific effects as well as scenario-specific multi-path fading characteristics are applied by the *MIMO Channel and Propagation Environment* model, before transmission status and overall delay are reported back to the emulation.

a) Medium Access: Virtual WLAN devices created with help of the Linux kernel module *mac80211_hwsim* only pass frames to and from the software MAC layer *mac80211*. Thereby, frames transmitted between emulated nodes on the same channel are simply copied. Even when relying on *wmediumd* and its mobility extension [5], there is no consideration of realistic channel effects, random back-off, or delays caused by consecutive frame retransmissions. Thus, apart from a complex physical-layer model, we also integrate our own models for carrier sense and the IEEE 802.11e Enhanced Distributed Channel Access (EDCA) protocol into *wmediumd*.

Originating from the IEEE 802.11e standard, EDCA is defined as mandatory default channel access mechanism for IEEE 802.11s networks [1]. It is based on classical CSMA/CA but supports traffic prioritization by using different access categories (AC) for video, voice, best effort, and background data. Practically, each WLAN device maintains a separate queue and back-off timer per AC. Without priority mapping, explicitly issued on higher layers, frames are sent as best effort traffic. Furthermore, for realistically evaluating IEEE 802.11s mesh networks in multi-channel operation, the consideration of collision domains is part of our EDCA model. Thereby, nodes on the same channel within signal detection range, as determined by carrier sense, form a collision domain and consider each other in the medium access protocol.

b) Mobility, Path Loss and Interference:
We adopted the mechanism for topology and mobility definition from the original *wmediumd* implementation and its extension by Illan et al. [4], [5]. As in the initial version, a two-dimensional area of arbitrary size is used for node placement. The mobility extension allows for the definition of a coordinate sequence per node that is executed in a step-wise manner. Its accuracy depends on the choice of distance and time intervals. In the original version, a minimum of 1 m distance and 1 s time steps could be configured. We increased both resolutions to floating point accuracy. The distance between nodes is then recognized to calculate the path loss (large-scale fading) for frame transmissions. The original *wmediumd* extension uses a simplified path loss calculation based on linear regression [5]. However, we replaced it with a more complex free-space path loss model that is commonly used in theory [14].

Depending on the collision domains, as determined by the *Medium Access* model, co- and adjacent-channel interference are calculated. This enables a realistic evaluation of multi-channel mesh networks and the influence of transmissions on overlapping channels. Whereas co-channel interference only depends on the spatial distance between nodes in the same collision domain, signal attenuation caused by transmissions on adjacent channels is additionally derived from the overlap of their 20 or 40 MHz wide spectral masks, according to the IEEE 802.11 standard [1].

c) MIMO Channel and Propagation Environment:
As most comprehensive extension to the original *wmediumd* simulator, we integrate our own complex physical-layer model that allows for the simulation of WLAN networks based on the IEEE 802.11a/g/n standards using OFDM multi-carrier modulation. We developed and evaluated our initial channel model using MATLAB [14], [15], featuring comprehensive tool-sets in this domain, before rewriting the model as a C implementation and integrating it into *wmediumd*.

Our model supports the 2.4 and 5 GHz frequency bands with channel widths of 20 and 40 MHz. Moreover, it implements all IEEE 802.11n modulation and coding schemes (MCS), including legacy (IEEE 802.11a/g) and high throughput (HT) modes with long or short guard intervals (GI). Common effects of real multi-carrier communication systems are considered, such as the influence of antenna gains and additive noise on the signal-to-noise ratio (SNR), as well as inter-symbol-interference (ISI). Different antenna configurations can be configured to support IEEE 802.11n transmit/receive diversity and spatial multiplexing techniques. Currently, an upper limit of two antennas per mesh node persists, which allows for the evaluation of 2x2 MIMO systems.

For the evaluation of small-scale fading effects, we integrated official environment models of the WINNER-II project [16]. This model set includes line-of-sight/non-line-of-sight multi-path propagation gains for 18 indoor and outdoor scenarios (urban, suburban, rural) that are valid for a frequency range of 2 to 6 GHz, thus including the 2.4 and 5 GHz bands.

V. Evaluation

We demonstrate the physical-layer simulation model of *ViPMesh* using two example scenarios. We run simulations for a single transmission link, applying the physical-layer models, i.e., simulation steps two and three as depicted in Section IV-D. Our extended *wmediumd* simulator is initialized with a configuration file, containing, e.g., the network topology, mobility definitions, propagation environment, channel parameters, or antenna configurations, applied within the models.

We analyze our channel model by generating a data stream between a simulated transmitter and receiver and calculating the resulting bit error rate (BER). It denotes the percentage of erroneous bits in the amount of data received and is commonly used as quality indicator for digital transmission paths.

In a first scenario, we vary the transmitter-side signal-to-noise ratio (SNR) in 1 dB steps, between 0 and 40 dB. For each setting, an overall of 10,000 pseudo frames is generated as input for the channel model, containing 500 bytes of random payload. Thereby, the average BER over all frames is calculated. We simulate a 2.4 GHz transmission channel having a width of 20 MHz and long guard interval (GI) with high throughput (HT) mode disabled, resulting in 48 data sub-carriers per OFDM symbol. We apply a flat-fading multi-path propagation environment model, originating from the WINNER-II project [16], comprising some strong line-of-sight (LOS) and many weak non-line-of-sight (NLOS) paths. We further configure up to two antennas per node, used as transmit and/or receive chains, and evaluate different mappings, including MIMO variants:

- SISO (1 transmit/receive chain, 1 stream)
- SIMO (1 receive, 2 transmit chain(s), 1 stream)
- MISO (2 receive, 1 transmit chain(s), 1 stream)
- DIV-MIMO (2 receive/transmit chains, 1 stream)
- SP-MIMO (2 receive/transmit chains, 2 streams)

The variants SIMO (transmit diversity), MISO (receive diversity), and diversity MIMO (combination) transmit and/or receive the same spatial stream redundantly. Contrary, spatial multiplexing (SP-MIMO) transmits and receives two independent spatial data streams. To verify the ability of our model to represent the dependence between modulation order and required SNR, we apply different IEEE 802.11n modulation and coding scheme (MCS) indexes, denoting different modulation variants. Thereby, the given MCS indexes correspond to the single- and dual-stream variants.

- MCS 1 (9): QPSK with coding rate 1/2
- MCS 3 (11): 16-QAM with coding rate 1/2
- MCS 6 (14): 64-QAM with coding rate 3/4

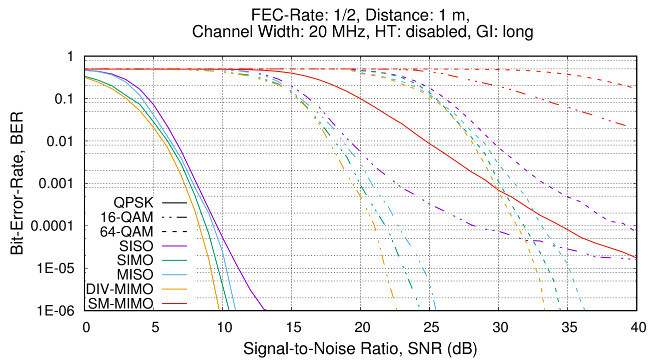

Fig. 5: BER depending on SNR for different antenna configurations and modulation orders

Figure 5 shows, in logarithmic scale, the BER between transmitter and receiver in 1 m distance for the different antenna configurations and OFDM modulation orders, depending on the SNR. For a 0 dB SNR (equal signal and noise power), BER is 50 %. As expected, a decreasing BER can be observed for an increasing SNR. When using higher modulation orders, leading to more bits per symbol and higher theoretical data rates, also a higher SNR is needed as symbol transmission becomes more error-prone. As shown in Figure 5, 64-QAM (6 bits/symbol) needs an SNR of 25 dB to achieve a BER of 10 % whereas 16-QAM and QPSK only require an SNR of 15 dB and 3 dB in the best case, respectively.

Additionally, the differences between the antenna configurations become clearly visible, confirming theoretical expectations. Thereby, SIMO and MISO allow for diversity, i.e., transmitting or receiving the same spatial stream redundantly whereas DIV-MIMO combines both diversity variants. This results in an SNR gain, especially when applied in multipath propagation environments. Thus, the diversity variants outperform SISO, respectively. On the other hand, SM-MIMO is capable of transmitting two independent spatial streams with a theoretical doubling in data rate. However, for the correct signal detection at the receiver it needs the highest SNR by far, compared to all diversity variants and SISO. In general, spatial multiplexing benefits from dominating NLOS paths to distinguish received streams. Thus, the applied propagation environment, comprising strong LOS paths, is not particularly suitable for SM-MIMO, as confirmed by our simulations.

Figure 6 shows the results of a similar scenario for an increasing distance between transmitter and receiver, this time

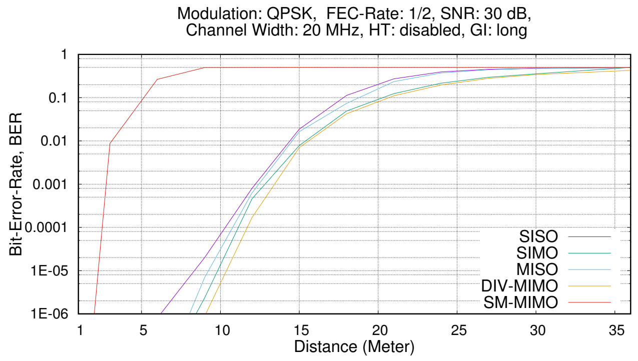

Fig. 6: BER depending on inter-node distance for different antenna configurations and QPSK modulation

using only QPSK modulation. Thereby, distance is increased in 3 m steps from 1 to 36 m. Transmitter-side SNR is configured to 30 dB, considering environmental noise. All other configurations are kept as described previously. Results show an increasing BER with increasing distance. This demonstrates the ability of our simulation model to represent the dependence between inter-node distance and SNR due to free-space path loss (large-scale fading).

VI. Related Work

We further compare our concept with current approaches in research to position it with regard to the state-of-the-art.

In [17], the authors propose an approach to include real implementations of TCP/IP stacks and applications in a wireless network simulation. By using virtual machines (VMs) that are controlled by the event calendar of the network simulation process, a continuous time wall clock is not required but event-driven simulations are run. Thereby, simulations can be run that are slower or faster than real-time. For the implementation of *VipMesh*, we adopt this concept of time decoupling. In [2], the authors extend their approach and present VMSimInt, which is a framework that integrates VMs into a network simulation tool to provide realistic OS behavior. Thereby, the focus is on providing a realistic TCP implementation. In their approach, each node is placed in its own VM, so the number of nodes corresponds to the number of required virtual machines. As opposed to our approach of isolating all nodes in a single VM, using lightweight nested container virtualization, the concept of [2] does not scale very well in terms of memory requirements as well as communication and control overhead that increase significantly with any additional node and VM.

A virtual time system for OpenVZ-based network emulations is presented in [18]. The authors modify OpenVZ and its schedulers to be able to provide VMs each with their own virtual time, running on a single OS. As opposed to heavyweight systems like Xen whose VMs contain both OS and application, this approach scales better with an increasing number of VMs. VMs and their virtual times are managed by a control application running on the host OS and simulating a network of choice. The approach currently solely features container-based virtualization for Linux but prospectively the authors aim at developing a virtual time system for QEMU to support a larger number of platforms. In contrast, we combine Linux containers with a time-controlled QEMU-based system virtualization. Moreover, we integrate comprehensive physical-layer simulation models for IEEE 802.11s mesh networks.

The work [19] addresses the problem of time divergence in hybrid network emulation by introducing a system called TimeSync that uses discrete-event simulation time to control and synchronize time advance on VMs. The core idea of TimeSync is to create a simulator-driven virtual timeline in the VMs participating in emulation. Although the core idea is similar to our approach, the focus of TimeSync is different as it uses stationary nodes connected by a wired Ethernet network rather than supporting IEEE 802.11(s) networks consisting of both stationary and mobile nodes.

An approach working with modified virtual clock requests is presented in [20]. The developed time dilatation program TimeKeeper redirects the access of Linux containers from the host clock to modified virtual clocks. These clocks can be controlled with TimeKeeper and therefore it both controls the time dilatation in the isolated virtual environments and the synchronization between these environments. This concept can basically be integrated into a combined emulation/simulation environment as it allows for the resource-efficient virtualization of nodes. However, our tests revealed that a Linux network namespace or container does not utilize a fully isolated network stack and hence TimeKeeper does not completely decouple the virtual times from the wall clock. Consequently, this approach shows severe limitations regarding the timing precision when integrating it into our architectural concept.

In summary, *ViPMesh* improves the state-of-the-art by a first approach that complements a real protocol stack, including the IEEE 802.11s reference implementation as applied in practical systems, with a set of comprehensive simulation models that allow for the early design evaluation of real applications and algorithms in WLAN mesh setups with IEEE 802.11n MIMO techniques, multi-channel operation, and mobility.

VII. Conclusion

We present *ViPMesh*, a virtual prototyping framework for WLAN mesh networks based on IEEE 802.11s and its Linux reference implementation. ViPMesh relies on WLAN interface emulation and QEMU-based system virtualization with nested container isolation to support the early design analysis of real applications on top of an unmodified network stack. Thus, despite our particular focus on IEEE 802.11s, this approach also allows for the evaluation of common infrastructure-mode WLAN setups or any other MAC-layer variant, as configurable under Linux. Adopting an alternative time source approach for QEMU, ViPMesh acts as discrete-event simulator. Furthermore, it integrates comprehensive medium access, channel, and environment models with support for interference effects, IEEE 802.11n MIMO, multi-channel operation, and device mobility. The proof-of-concept implementation of ViPMesh is evaluated in example scenarios, demonstrating its physical-layer model. To further calibrate our simulation models, comparative measurements will be conducted in corresponding real-world setups, using a 40-node test bed at our institute. Our MIMO channel model also allows for a straight-forward extension to integrate, e.g., modulation and coding schemes added by the recent IEEE 802.11ac standard.

Acknowledgment

The authors would like to thank the German Research Foundation (DFG), RTG 1424 (MuSAMA) and research fellowship, GZ: DA 1687/2-1, for their financial support.

References

[1] "IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, 2012.

[2] T. Werthmann, M. Kaschub, M. Kühlewind, S. Scholz, and D. Wagner, "VMSimInt: A Network Simulation Tool Supporting Integration of Arbitrary Kernels and Applications," in *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014, pp. 56–65.

[3] "open80211s," 2016. [Online]. Available: http://open80211s.org/

[4] "Cozybit Wireless Medium Simulator (wmediumd)," 2016. [Online]. Available: https://github.com/cozybit/wmediumd

[5] A. Martínez Illán, "Medium and mobility behaviour insertion for 802.11 emulated networks," Master's thesis, Universitat Politècnica de Catalunya, 2013.

[6] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *Proceedings of the USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 41–41.

[7] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*, March 2015, pp. 171–172.

[8] "ns-3," 2016. [Online]. Available: https://www.nsnam.org/

[9] P. Owczarek and P. Zwierzykowski, "Review of simulators for wireless mesh networks," *Journal of Telecommunications and Information Technology*, no. 3, p. 82, 2014.

[10] "OMNet++," 2016. [Online]. Available: https://omnetpp.org/

[11] S.-Y. Wang and Y.-M. Huang, "NCTUns distributed network emulator," *Internet Journal*, vol. 4, no. 2, pp. 61–94, 2012.

[12] R. Barr, "SWANS - Scalable Wireless Ad Hoc Network Simulator," 2004. [Online]. Available: http://jist.ece.cornell.edu/docs.html

[13] R. Russell, "Virtio: Towards a De-facto Standard for Virtual I/O Devices," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 95–103, Jul. 2008.

[14] B. Sklar, *Digital communications: fundamentals and applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[15] Y. S. Cho, J. Kim, W. Y. Yang, and C. G. Kang, *MIMO-OFDM Wireless Communications with MATLAB*. Wiley Publishing, 2010.

[16] "WINNER II Channel Models," September 2007. [Online]. Available: http://www.ist-winner.org/WINNER2-Deliverables/D1.1.2v1.1.pdf

[17] T. Werthmann, M. Kaschub, C. Blankenhorn, and C. M. Mueller, "Approaches for evaluating the application performance of future mobile networks," *European Cooperation in the Field of Scientific and Technical Research, COST IC1004 TD (11)*, vol. 1038, 2011.

[18] Y. Zheng and D. M. Nicol, "A Virtual Time System for OpenVZ-Based Network Emulations," in *Principles of Advanced and Distributed Simulation (PADS), 2011 IEEE Workshop on*, June 2011, pp. 1–10.

[19] F. Sultan, A. Poylisher, C. Serban, J. Lee, R. Chadha, C. J. Chiang, K. Whittaker, C. Scilla, and S. Ali, "Timesync: Virtual time for scalable, high-fidelity hybrid network emulation," in *IEEE MILCOM*, 2012.

[20] J. Lamps, D. M. Nicol, and M. Caesar, "TimeKeeper: A Lightweight Virtual Time System for Linux," in *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM PADS '14. New York, NY, USA: ACM, 2014, pp. 179–186.