Simulation

Simulation: Transactions of the Society of Modeling and Simulation International
0(00) 1–16
© The Author(s) 2010
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0037549710373909
sim.sagepub.com

VirtualMesh: an emulation framework for wireless mesh and ad hoc networks in OMNeT++

Thomas Staub, Reto Gantenbein and Torsten Braun

Abstract
Wireless Mesh Networks (WMNs) have proven to be a key technology for increased network coverage of Internet infrastructures. The development process for new WMN protocols and architectures is typically split into evaluation by network simulation and testing of a prototype in a test-bed. Testing in simulation often requires the developer to write software that is not directly portable to test-beds, whereas pure prototype testing on real hardware is time consuming and expensive. Irrepressible external interferences can occur, which make debugging difficult. Moreover, the test-bed usually supports only a limited number of test topologies and sites. Finally, mobility tests are impractical. Therefore, we propose VirtualMesh as a novel testing architecture, which can be used before evaluating in a real test-bed. It provides instruments to test the real communication software, including the network stack inside a controlled environment. VirtualMesh has been implemented by capturing real traffic through a virtual interface at the mesh nodes. The traffic is then redirected to the network simulator OMNeT++. In our experiments, VirtualMesh has proven to be scalable, to have minimal influence on throughput and to introduce only negligible delays (less than 0.4 ms per hop). Hence, it is a valuable tool for protocol and application developers to test their software prior to the final deployment.

Keywords
integration of real nodes in a simulated environment, network emulation, network simulation, OMNeT++, pre-deployment testing, wireless emulation

1. Introduction

Wireless Mesh Networks (WMNs) have become one of the key technologies for providing increased network coverage of Internet infrastructures. Their simple, cost-efficient deployment with self-configuration facilities makes them a valuable alternative to wired networks to increase network coverage.[1] Therefore, WMNs are in the focus of current research. Several research and city WMNs already exist.[2–4] Currently, WMNs are evolving from pure research networks to carrier-grade communication infrastructures, which require extensive pre-deployment testing.

For the commercial utilization of WMNs, new communication protocols as well as new customer services have to be developed. The development process in WMNs is typically split into evaluations by simulations and testing a real prototype in a test-bed. Firstly, protocols and architectures are implemented and evaluated in a network simulator. Afterwards, a prototype is implemented on the target platform, such as Linux, and tested inside a test-bed before deployment in the real network. Simulation provides most flexibility in testing. Different and large-scale experiments, as well as experiments with mobility of devices and users, are possible. Thus, the focus here can be set on testing and debugging the functionality of the proposed protocols. Unfortunately, simulation models cannot cover all influences of the operating system, the network stack,

Institute of Computer Science and Applied Mathematics, University of Bern, Switzerland.

Corresponding author:
Thomas Staub, Institute of Computer Science and Applied Mathematics, University of Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland
Email: staub@iam.unibe.ch

the hardware and the physical environment due to complexity constraints. Therefore, the transition from simulation models to the deployable solution remains challenging. Testing the prototype in a test-bed during the implementation process is time consuming, costly and very limited in test scenarios. Due to economical reasons, the scale of the test-beds is limited and they are often not deployed in isolated environments, which limit reproducibility. Interferences with existing networks are possible and irrepressible, which makes the debugging of new protocols very challenging. Furthermore, the number of test topologies is limited and mobility tests are impracticable. Moreover, WMNs provide an enhanced testing challenge compared to simple wireless access networks. They support mobile users and high-throughput applications. Their architecture contains self-configuring and self-healing mechanisms, which have to be included in the tests. Cross-layer interactions have to be tested in a controlled environment without any irrepressible influences. Moreover, the tests have to cover the time and delay aspects of the real network stack. Not all these tests can be fully done in simulations, but it is also difficult to perform them in a test-bed.

We therefore propose to use the final operating software of the nodes, to replace the wireless interfaces with virtual ones and to emulate the physical medium for gaining more control in the development process. This substantially facilitates the testing process, as the real software stack may be evaluated within a controlled environment.

Our contribution is an emulation framework called VirtualMesh for WMNs, based on the network simulator OMNeT++. This framework offers enhanced evaluation of communication software written for real and virtualized nodes on top of an OMNeT++ simulation model. Communication software can be tested without any adaptations over an emulated network in OMNeT++. VirtualMesh uses real mesh nodes with a real network stack. It intercepts wireless traffic before transmitting it over the air and forwards it to a simulation model. This simulation model offers a vast flexibility in topologies and mobility tests. In addition, the scale of the test scenarios can be increased by host virtualization.[5]

The remainder of this paper is structured as follows. Firstly, we provide an overview of related work in Section 2. Then, we present the general architecture of VirtualMesh in Section 3, with its key concepts, packet interception and forwarding in Section 3.1 and processing of network traffic by a simulation model in Section 3.2. Besides details about the simulation model, Section 3.2 describes the integration of real traffic into the simulator. The individual procedures used in VirtualMesh are illustrated in Section 3.3. Afterwards, we provide an evaluation of VirtualMesh in Section 4 and show the value of VirtualMesh for pre-deployment tests. Finally, we present our conclusions in Section 5.

The work presented in this paper extends the VirtualMesh architecture and the corresponding evaluation published in Staub et al.[6] The internal message flow has been redesigned to reduce the latencies introduced by VirtualMesh and to support dynamic scenarios with nodes joining and leaving. Besides a detailed and illustrated description of the VirtualMesh architecture, an enhanced evaluation is presented.

2. Related work

Emulation is valuable for network research and protocol development. It can approximate the real environment more accurately than pure simulation. In Ivanov et al.[7] the authors validated the wireless model in the network simulator ns-2 by comparing measurements of a real network setup with an emulated and simulated network. They concluded that with a proper parameterization the simulation model can approximate the real network, but some aspects, such as delays introduced by hardware and the operating system, cannot be considered in the simulation. Therefore, their emulated network provides results that matched the real measurement more accurately than the simulation.

In the following we discuss several approaches for wireless network emulation and compare them to VirtualMesh.

The combination of node virtualization and network emulation used in VirtualMesh has been proposed by Engel et al.,[5] Krop et al.,[8] and Zimmermann et al.[9] These three approaches are explained in more detail in the following.

The approach presented in Engel et al.[5] tries to integrate the behaviour of the real network stack and the operating system into the testing process by using virtualized hosts connected through an emulation framework. The virtual hosts are running an L4 microkernel on top of a real-time kernel. To integrate the wireless network behaviour, the hosts are connected by the 802.11b network emulator MobiEmu.[10] Like in VirtualMesh, the wireless interface driver has been modified to communicate with the emulator instead of the physical interface, but keeping the interface to the applications unaltered. A drawback of the approach is inherited by the use of MobiEmu; the communication is either possible without errors or not at all. In comparison to VirtualMesh, MobiEmu does not model any communication errors. Unfortunately, no results about the accuracy of the setup are available.

UMIC-Mesh[9] is a hybrid WMN test-bed. Besides a test-bed with real wireless mesh nodes, UMIC-Mesh provides virtual nodes by using XEN[11] virtualization.

The virtual nodes are interconnected by a combination of the advanced networking features of the Linux kernel. This includes packet filtering for controlling the communication between the nodes. The virtual network is only intended for software development and functionality validation. Therefore, the behaviour of the wireless medium has not been modelled in this approach.

JiST/MobNet[8] provides a comprehensible Java framework for the simulation, emulation and real-world testing of a wireless ad hoc network. It allows the running of the same tests independently of the platform and abstraction level. MobNet is a wireless extension on top of the Java in Simulation Time (JiST) simulator. The drawback of this approach is that most communication software and network protocol stacks are not written in Java and therefore a further transition to a real-world system may be necessary afterwards.

Another approach for testing real implementations in a very flexible network is provided by the ORBIT test-bed.[12] It provides a configurable indoor radio grid for controlled experimentation and an outdoor wireless network for testing under real-world conditions. The indoor radio grid offers a controlled environment as an isolated network, in which background interferences can be injected. Although the 20×20 grid of nodes offers a large variety of different topologies, it can be too restricted and mobility tests are even more limited. Furthermore, the scarce ORBIT resources may be not available for all experiments.

The network test-bed Emulab[13] provides various experimentation facilities with advanced experiment management controls. For experiments with wired networks, network nodes run standard operating systems (FreeBSD, Linux and Windows XP) and communicate over an emulated network using virtual local area networks (VLANs) and the emulator Dummynet.[14] Emulab has been extended to the wireless domain[15] by an IEEE 802.11a/b/g test-bed. Several nodes with real wireless interfaces are deployed on the floors of an office building and can be integrated in an Emulab experiment scenario. Besides the lack of mobility support, the Emulab wireless test-bed suffers from limited repeatability due to the shared location in an office building with interferences from productive networks.

Limited mobility is supported in a further test-bed, named mobile Emulab.[16] However, currently, mobile Emulab is not suitable for IEEE 802.11-based networks. Small robots, whose movements can be remotely controlled through an Emulab control script, carry wireless sensor motes with 900 MHz radios. The current setup uses an IEEE 802.11b network for the remote control. This and the size of the test-bed room limit possible extensions of mobile Emulab for wireless LAN (WLAN) experiments. Moreover, the robots-based test-bed is a costly and scarce resource, similar to the ORBIT test-bed. In comparison, VirtualMesh is a cost-efficient test-bed as it is composed of standard computers and network components. The complete emulation of the wireless medium avoids interference problems and offers arbitrary network topologies, including mobility by a simple adaptation of the simulation model.

The wireless network emulator QOMET[17,18] converts the wireless scenario into a time-series of network states. This state description is then delivered to wired-network emulator Dummynet[14] in order to emulate the wireless link between end points. The end points are standard computers emulating the wireless nodes. The communication is sent over Dummynet using a wired network. QOMET provides repeatability and testing of real application software. As the normal operating system tools cannot change the wireless parameters of the network, QOMET is not suitable for testing software that influences the wireless interface of a node, which is possible in VirtualMesh due to the standard interface of the wireless devices.

Another interesting approach is a wireless emulator using a hardware channel simulator.[19,20] The unaltered network nodes are packed in radio frequency (RF)-shielded boxes and their radio interfaces are connected to the hardware channel simulator, which then emulates the signal propagation using a field-programmable gate array (FPGA). The channel simulator supports directional antennas and mobility. The system presented in Borries et al.[20] supports 15 nodes operating in a 2.4 GHz industrial, scientific and medical (ISM) band. The main advantage of a FPGA-based wireless emulator is the provided repeatability in combination with a real media access control (MAC) layer and a realistic physical layer supporting multipath fading. The main drawbacks are the costs and the limited number of nodes. VirtualMesh provides more flexibility due to the usage of commodity hardware.

Moreover, the integration of real network stacks inside a network simulator provides another mechanism for testing complex protocol behaviour. OppBSD[21] integrates the transmission control protocol/internet protocol (TCP/IP) stack of FreeBSD in the network simulator OMNeT++. The Network Simulation Cradle[22] project provides support for using the real network stacks of Linux, FreeBSD and OpenBSD with the network simulator ns-2. The integration of a real TCP/IP stack provides results that are closer to a real-world network. Nevertheless, both approaches do not support the testing of native unaltered applications, which is targeted by VirtualMesh.

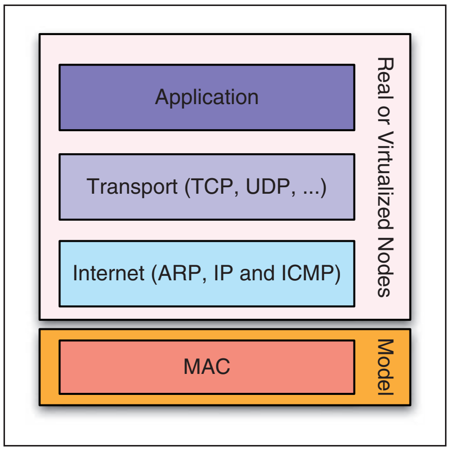
The newly developed network simulator ns-3[23] allows the integration of virtualized nodes running native applications and protocol stacks under the Linux operating system. The virtualized nodes in ns-3 are connected through a TUN/TAP device of the Linux kernel and a proxy node to the simulation. However, there is no support to modify device parameters of the simulation directly and dynamically by the virtualized nodes, especially for wireless devices. VirtualMesh shares a similar approach for traffic redirection and internal representation of virtualized nodes. However, in VirtualMesh, virtualized or real nodes can directly manipulate the wireless device parameters through usual system tools (e.g. iwconfig). In addition, dynamic changes of the parameters during the emulation are possible.

When injecting real network traffic into a network simulator, there is always the problem that the simulation may not keep pace with the real network. The simulation may be too slow. In order to cope with the problem of a simulator overload during network emulation, Weingärtner et al.[24] introduce the concept of synchronized network emulation. They replace real hosts with virtualized hosts using XEN. A central synchronizer component then controls the time flow of the virtual hosts by an adapted scheduler for XEN. It keeps them synchronized with the network simulator OMNeT++.[25] Synchronized network emulation represents a valuable extension to avoid scalability problems and could be integrated in VirtualMesh in the future.

3. VirtualMesh: concept and architecture

The main concept of VirtualMesh[6] is to intercept and redirect real traffic generated by real nodes to a simulation model, which then handles network access and the behaviour of the physical medium. The network stack is split into two parts as shown in Figure 1. The application, transport and Internet layer are handled by the real/virtualized node. At the MAC layer, the traffic is captured by a virtual network interface and then redirected to the simulation model. The simulation model calculates the network response according to the virtual network topology, the propagation model, the background interferences and the current position of the nodes. Only the MAC layer and the physical medium are simulated. All the other layers remain unchanged and work just as in a real test-bed of embedded Linux nodes.

VirtualMesh combines the advantages of real-world tests performed on embedded Linux systems with the flexibility and the controlled environment of a network simulator. The main advantages are: the real communication software is used, the real network stack is tested, the effects of temporary unavailable nodes can be evaluated, background traffic/interferences can be controlled and different mobility tests can be easily performed. The real implementation of the communication software can be tested. Accordingly, the behaviour of the Linux network stack is embedded in a controlled testing environment. There are no irrepressible influences on the experiments, such as interferences from neighbouring networks and power lines, steel structures of buildings or changing weather conditions. In addition, the underlying simulated network enables large-scale experiments. It supports changing topologies and different mobility scenarios. This makes automated testing of the real communication software with a high variety of scenarios possible.


Application
Transport (TCP, UDP, ...)
Internet (ARP, IP and ICMP)
Real or Virtualized Nodes
MAC
Model
Figure 1. Subdivision of the network stack to the real/virtualized nodes and the simulation model.

The general architecture of VirtualMesh is shown in Figure 2. It consists of an arbitrary number of computers hosting the simulation model and real or virtualized mesh nodes. A dedicated IEEE 802.3 Ethernet service network interconnects the nodes and the model. The wireless interfaces of the nodes are replaced by virtual interfaces, which communicate over the service network to the simulation model. Besides real nodes, the architecture supports virtualized hosts. Host virtualization is performed by XEN, but other virtualization techniques could be used too. Host virtualization provides additional scalability of the system. One standard server machine may hold up to ten virtual mesh nodes without any problem.

3.1. Traffic redirection: virtual interface and PacketModeller

Traffic interception/redirection at the MAC layer is the fundamental concept that VirtualMesh builds upon. In order to redirect wireless traffic from the virtual/real nodes to the simulation model, we replace the actual wireless device by a virtual interface in combination with the *PacketModeller* daemon. Our virtual wireless

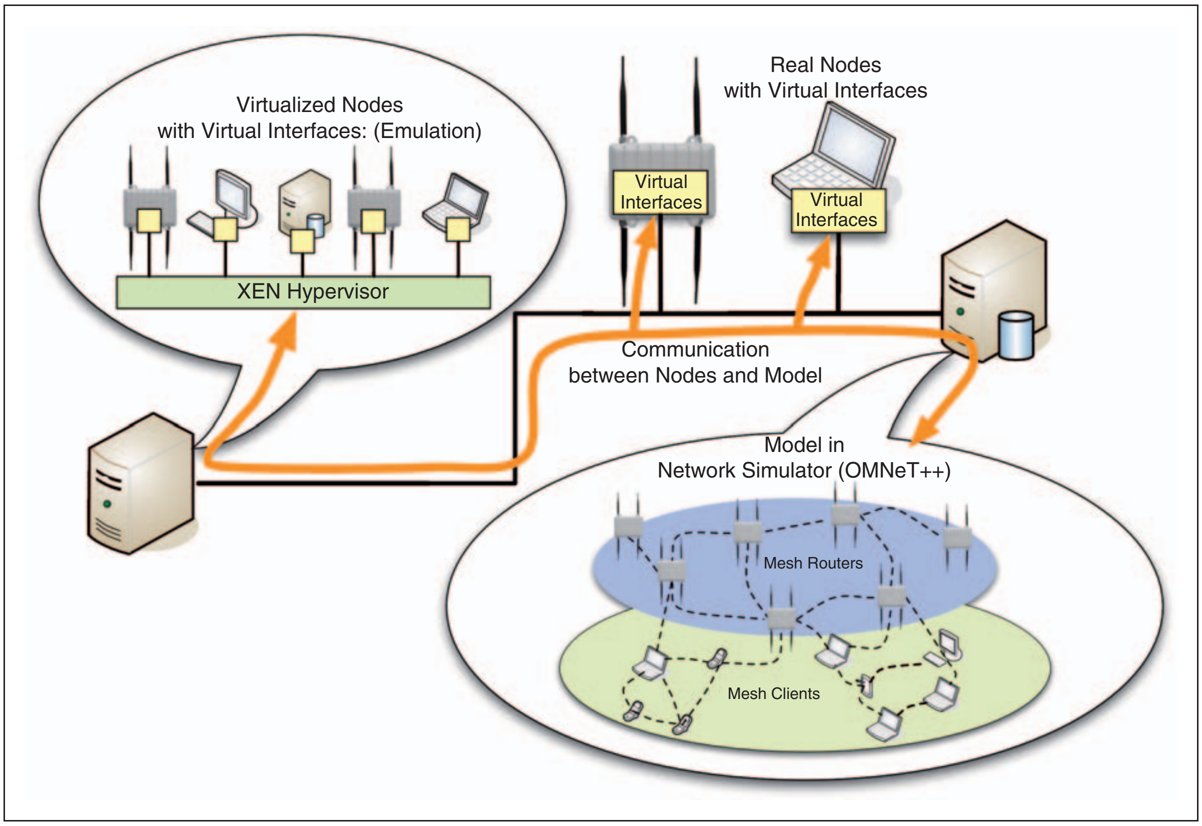


Figure 2. VirtualMesh architecture with real nodes, virtualized nodes and the simulation model.

device is built on top of the TUN/TAP device of the Linux kernel and is managed by a small user space virtual interface device daemon (*vifd*). Of course, this could have been implemented combined as a Linux kernel module, which should lead to a performance increase, but in favour of a broader applicability we decided against this solution. In our approach the testing environment is not bound to a specific out-of-tree kernel add-on and the daemon could even be ported to other operating system platforms, which offer a POSIX system interface and a TUN/TAP driver (e.g. *BSD, Darwin). The TUN/TAP device redirects any received network traffic as Ethernet frames to the user-space, where the *PacketModeller* takes care of them, while *vifd* is responsible for the configuration of the wireless parameters. On the other side, the *PacketModeller* reinjects the traffic that it receives from the simulation model to the Linux networking stack via the TUN/TAP device.

The configuration of the network device is performed at machine installation and adapted during the lifetime. A standard Linux network interface (see Figure 3(a)) is configured by net-tools or by the ip-route2 suite (i.e. by the commands *ifconfig* or *ip*). In addition, for wireless devices, wireless parameters such as wireless channel, operation mode, transmission power, Ready-to-Send/Clear-to-Send threshold and encryption are set by the wireless-tools (e.g. *iwconfig*) through the Wireless Extension application programming interface (API) of Linux. Due to the use of the kernel's TUN/TAP driver, our virtual device behaves the same as any Linux network device. Hence, no changes in the network configuration itself are required. Furthermore, the wireless parameters of our virtual interface can be set by a patched version of wireless tools, such as *iwconfig* (see Figure 3(b)), which then sets the parameters in our device daemon *vifd*.

The user space daemon *PacketModeller* receives all packets transmitted to the virtual interface and encapsulates them in new packets, which are sent to the host running the simulation model. In the opposite direction, the *PacketModeller* is listening on a user datagram protocol (UDP) port for packets coming from the simulation model. These packets are then decapsulated and original Ethernet frames are injected back into the network stack via the virtual interface, which then passes them to the application (see Figure 4, numbers correspond to the individual steps). This way, the virtual interface, the *PacketModeller*, and the simulation model process the complete wireless traffic of the node.

Figure 4 shows the packet flow from the application at source node *S* to destination node *D*. Both nodes are

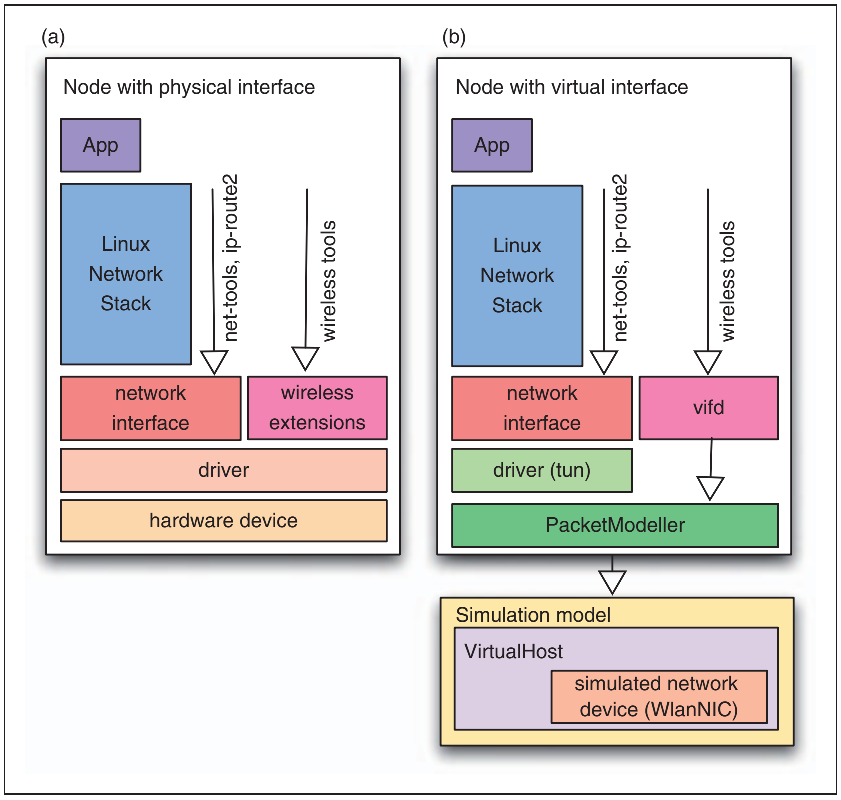


Figure 3. Node with Linux network stack and (a) a real network interface or (b) our virtual network interface (*PacketModeller*) communicating with the OMNeT++ simulation model.

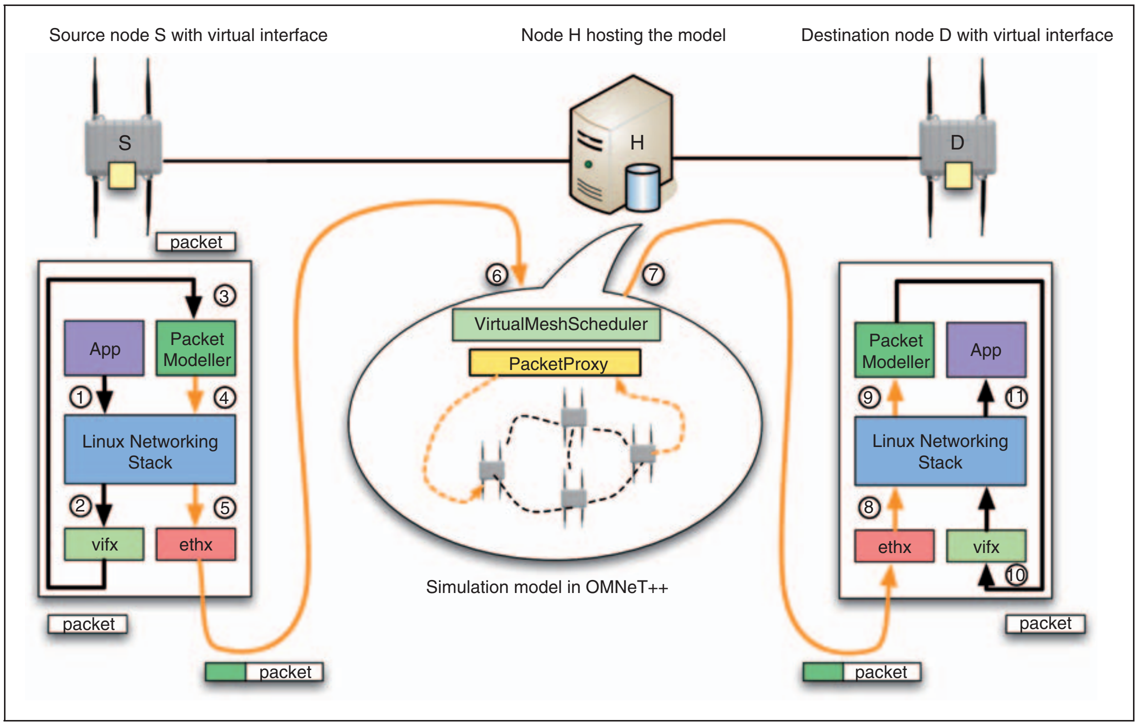


Figure 4. Packet flow between two nodes interconnected by the OMNeT++ simulation model.

connected to the simulation model on host *H*. The application at node *S* sends the packets to the Linux network stack (1), where they are intercepted by the virtual wireless interface *vifx* (2). The original Ethernet frames are then redirected to the *PacketModeller* (3), which encapsulates them in new packets (4). These packets are transmitted through the Ethernet interface *ethx* (5) to the simulation model on host *H* (6). At host *H*, the packets are fed into the simulation model (described later in Section 3.2). After processing in the simulation model, the resulting packets are encapsulated again and sent to their destination node *D* (7). There, the packets are received via the Ethernet interface *ethx* (8) and the *PacketModeller* (9). The *PacketModeller* decapsulates the packets and injects them back into the network stack via the virtual interface *vifx* (10). Finally, the application at node *D* receives the packets (11). Packet redirection is fully transparent for the applications and the network stack.

For accurate simulations, the model needs to incorporate several additional static and dynamic parameters describing the external nodes and the current configurations of their wireless interfaces. Static parameters (e.g. IP address and listening port of the *PacketModeller*) are set at startup of the node. The *PacketModeller* has to register itself at the model by a *REGISTRATION* message (see Figure 5). This message contains a sequence number (msg id), the host identification (unique id, e.g. 000b6bdbe502), the host name (e.g. node01), the infrastructure IP address (IPv4 or IPv6), the port where the *PacketModeller* is listening for incoming traffic and the number of interfaces. It further contains the initial values of dynamic parameters, such as interface name, MAC address and index, as well as wireless parameters (e.g. channel, transmission power, MAC level retries and receiver sensitivity) for each virtual interface. The *PacketModeller* sends the *REGISTRATION* message just after startup of the node. This *REGISTRATION* message is then retransmitted if not acknowledged by the model through an *ACK* message within 10 seconds. After successful reception of the acknowledgement, the node can start transmitting its wireless traffic to the model. Upon node registration, the model created an internal representation of the external node (*VirtualHost*). If any dynamic wireless parameters, such as the current communication channel and transmission power, have changed on the node, a *CONFIGURATION* message is sent to the model. The *CONFIGURATION* message includes the host identification, the index of the interface with the changes, and all the changed parameters of the interfaces as type/value tuples. The model is, therefore, supplied with these parameters and can calculate the simulation behaviour.

Regular network traffic between the nodes is sent as *DATA* messages, which are illustrated in Figure 5. A *DATA* message contains the intercepted Ethernet frame, its size and the host identification. In VirtualMesh, the management network uses a higher

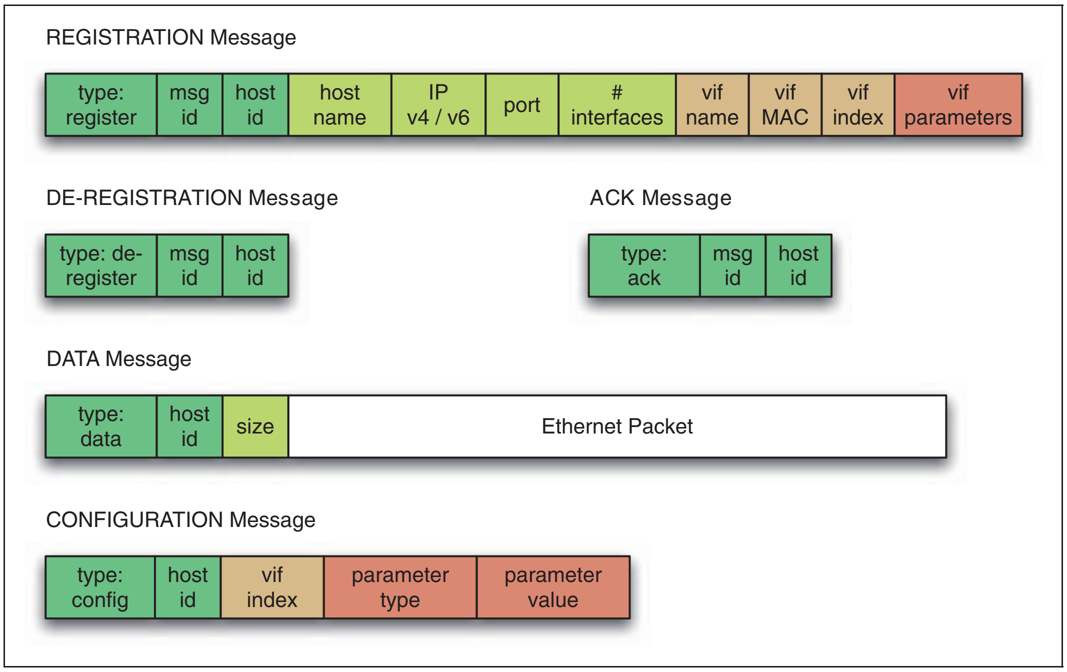


Figure 5. Message format to communicate with the model: node registration, data transmission and configuration.

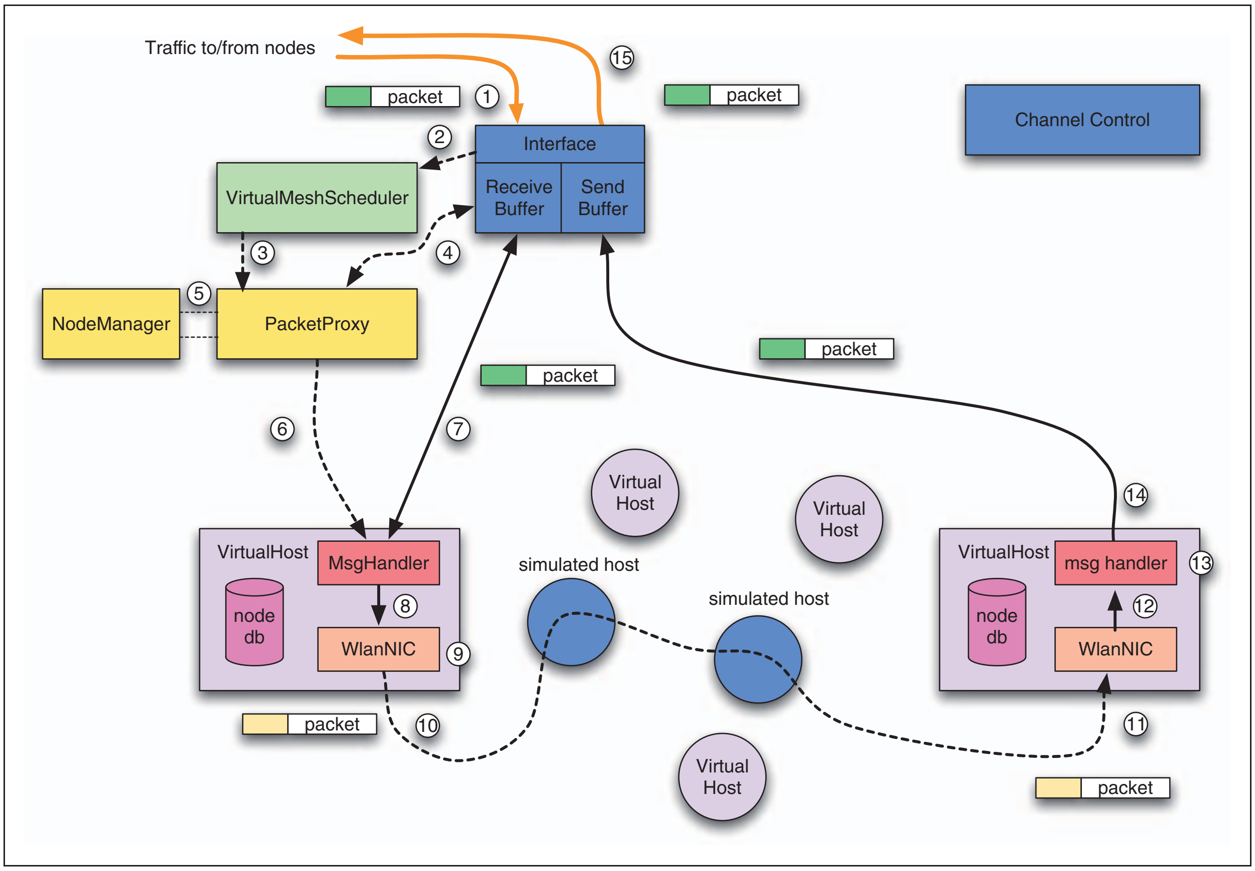


Figure 6. Packet flow inside the simulation model *WlanModel*.

maximum transfer unit (MTU) to compensate for the overhead of these *DATA* messages and guarantee the standard MTU for the data traffic coming from the nodes. As usually at least 1 Gbps interconnection is used, MTU sizes up to 9000 bytes are possible. At startup, the *PacketModeller* checks correctness of MTU sizes of the management network.

3.2. Simulation model

Our simulation model is called *WlanModel* and has been written as a module for the network simulator OMNeT++.[25] It receives traffic coming from external nodes, calculates the system's response and then sends the processed packets back to external nodes. It consists of the modules *VirtualMeshScheduler*, *PacketProxy*, *NodeManager*, and several *VirtualHosts*, including *MsgHandler* and the INET *Ieee-80211NicAdHoc* stack (*WlanNIC*). INET's ChannelControl and MobilityModels model the wireless network behaviour. Figure 6 shows the individual components, which are described in more detail in the following.

The main components of the *WlanModel* are the *PacketProxy* and the *VirtualMeshScheduler*. The *VirtualMeshScheduler* handles the simulation internal message scheduling including in-/outcoming network traffic. It listens on UDP port 2424 for incoming packets (as depicted in Figure 6) from the *PacketModeller* daemons of external nodes. Upon packet reception (1), the new packet is stored in the interface receive buffer, the *VirtualMeshScheduler* is informed (2) and schedules a notification message with the reception time for the *PacketProxy* module (3) in the message queue of the simulation. This message queue is then processed step-by-step, ensuring that the required timing constraints are met.

As soon as the *PacketProxy* module gets a notification message (3), it processes the received messages coming from outside the simulator (4). If a new external node registers its presence to the model, the *PacketProxy* calls the *NodeManager* (5), which acknowledges the reception of the *REGISTRATION* message. The *NodeManager* is responsible for the administration of external nodes inside the simulation model. If the registering node does not already exist in the simulation model, the *NodeManager* creates a new instance of *VirtualHost* and saves all node attributes (hostname, infrastructural IP address, listening port) to the node database of the *VirtualHost*.

After successful initialization, the *VirtualHost* acknowledges its presence to the external node by an *ACK* message.

Upon *DATA* message reception (4) the *PacketProxy* first checks whether the sending node has already registered at the *NodeManager* (5). If no registration exists, the packet is dropped immediately. Otherwise, the *PacketProxy* notifies the *MsgHandler* of the corresponding *VirtualHost* (6). The *MsgHandler* processes the packet (7). The encapsulated original Ethernet frame is included in a new *RAWEtherFrame* packet that is then transmitted to the *WlanNIC* of the *VirtualHost* (8). The *WlanNIC* uses the current wireless parameters from the node database for the transmission to the next node (9). Changes of the wireless parameters of external nodes are propagated to the simulation model by the transmission of a *CONFIGURATION* message. The *PacketProxy* is only involved in the processing of incoming traffic to the simulator.

The external nodes are modelled as *VirtualHosts*. A *VirtualHost* is a compound module of OMNeT++, i.e. it does not contain any message processing logic but simply adds a logical interface to a *MsgHandler* module, the node database and the existing *WlanNIC* module. The *MsgHandler* is called by the *PacketProxy* to process a new *DATA* message (6,7). It extracts the original Ethernet frame and generates a new *RAWEtherFrame* (8). The *RAWEtherFrame* is then further processed through the *WlanNIC*, which is the *Ieee80211NicAdhoc* model of the INET framework (9). Henceforth, the existing IEEE 802.11 model implementation of INET takes care of the packet (10) until it has been received again by a *VirtualHost* module (11). The *WlanNIC* checks whether the packet belongs to this node by checking the MAC addresses. If yes, it is forwarded to the *MsgHandler* (12). Otherwise, it is omitted. The *MsgHandler* generates a new *DATA* message that includes the packet (13). When the *Interface Send Buffer* gets a *DATA* packet from the *MsgHandler* (14), the packet is finally forwarded over the system network (15) to the external node.

3.3. Node registration, node de-registration, packet transmission, packet reception and the configuration process

In the following, the different processes in VirtualMesh are shown step-by-step covering the communication between the real node and simulation model, as well as the communication inside the simulation model. Actually, five processes exist in VirtualMesh. Firstly, the external node has to register itself at the simulation model (node registration). It can also cancel its registration within the model afterwards (node de-registration). When successfully registered, the external node can transmit packets (packet transmission) to its representation in the simulation model. After packet processing inside the simulated network, an internal representation of a node receives the packet and then transmits it to the connected external node (packet reception). By the transmission of a *CONFIGURATION* message, the external nodes can modify their interface parameters (node configuration). The numbers in the brackets (4.x and 6.y) reflect the steps in Figure 4 and Figure 6, respectively. In addition, node registration, de-registration, and configuration are illustrated in Figures 7–9.

3.3.1. Node registration

1. The node with a VirtualMesh interface boots. The configuration of the virtual interface contains the IP address and the port of the simulation model.
2. The node's *PacketModeller* sends a *REGISTRATION* message to the model (4.4–4.6).
3. The simulation model adds the node to the *NodeManager* and replies with an acknowledgement (6.1–6.5).
4. The *NodeManager* creates a new *VirtualHost*. The node database of the *VirtualHost* is initialized with the values of the *REGISTRATION* message. This includes host name, the infrastructural IP address and the listening port of the *PacketModeller*, and the number of interfaces and their configuration (name, MAC, transmission power, MAC level retries, receiver sensitivity, etc.).
5. Positions and mobility patterns of the *VirtualHost* have to be configured in advance by the OMNeT++ setup file and are therefore already present inside the simulation model.
6. Upon reception of the acknowledgement, the node is registered and can send/receive traffic to/from the simulation model.

3.3.2. Node de-registration

1. A shutdown of the *PacketModeller* daemon, e.g. when rebooting the external node, triggers the transmission of a *DE-REGISTRATION* message. It forces the *VirtualHost* to leave the emulated network (4.4–4.6).
2. Upon reception of a *DE-REGISTRATION* message, the *PacketProxy* invokes a node removal by the *NodeManager* (6.1–6.5).
3. The *NodeManager* removes the corresponding *VirtualHost*.

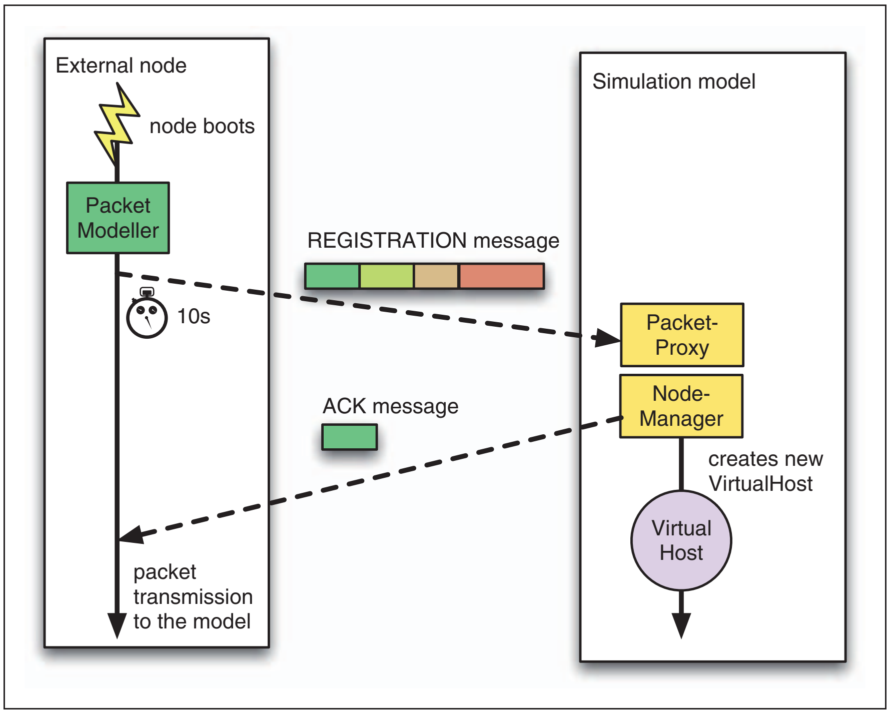


Figure 7. Node registration.

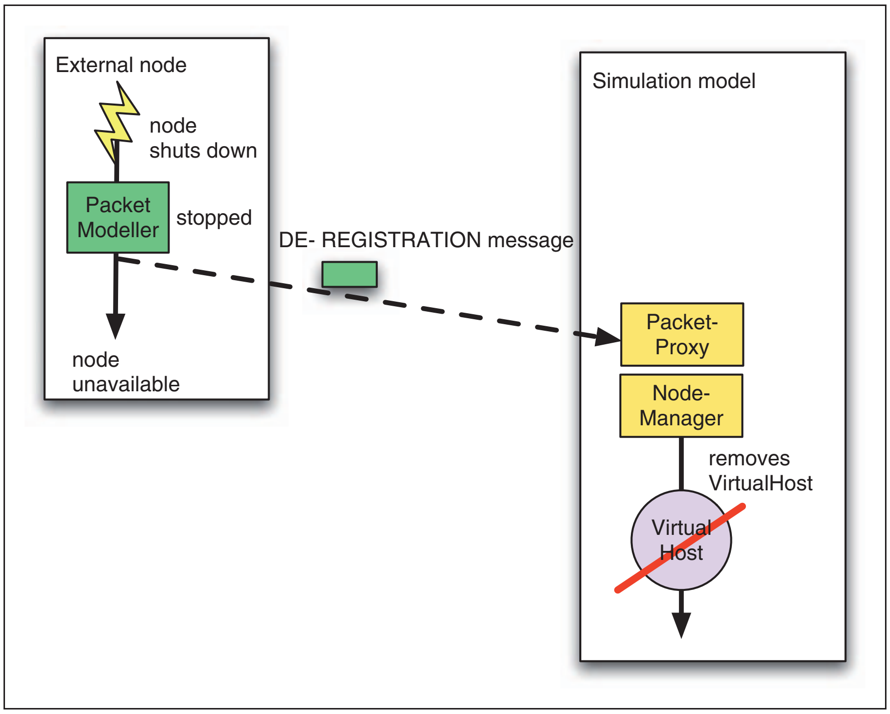


Figure 8. Node de-registration.

Node registration and de-registration allows the emulation of dynamic networks. Nodes can join and leave the network. They are automatically added and removed, respectively, from the simulation model. For example, this can be used to test a configuration and management framework. The effects of rebooting nodes or unavailable nodes at configuration time can be evaluated.

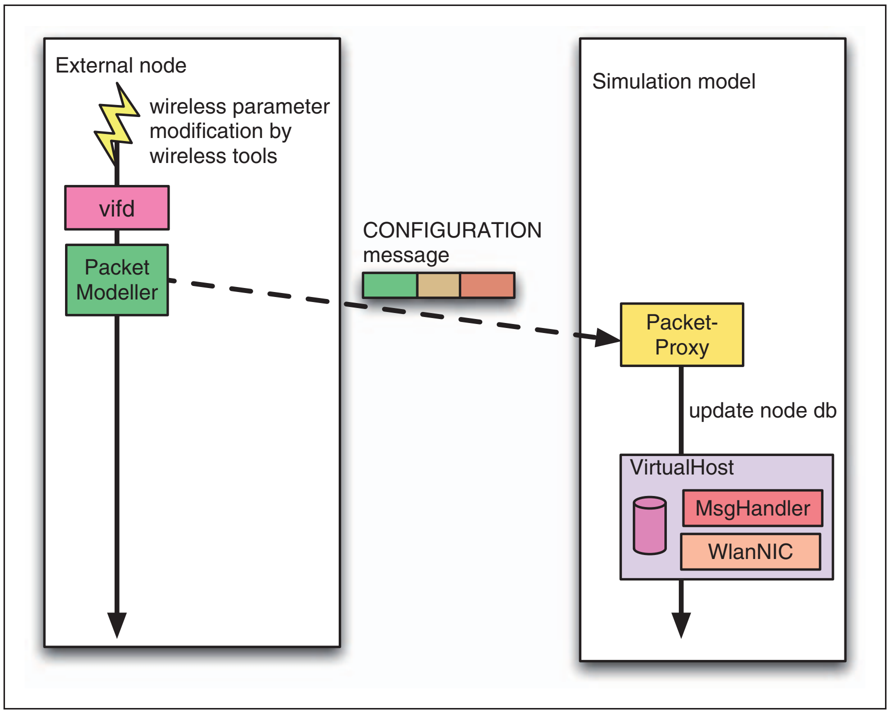


Figure 9. Node configuration.

3.3.3. *Packet transmission by an external node*

1. The source application at the node sends a packet to the virtual interface *vif* (4.1, 4.2).
2. The *PacketModeller* encapsulates this packet and then redirects it as *DATA* message to the simulation model (4.3–4.6).
3. The *DATA* message is received by the simulation model and stored in the *Receive Buffer* (6.1).
4. The *VirtualMeshScheduler* is notified and schedules the notification message to the *PacketProxy* (6.2, 6.3).
5. The *PacketProxy* checks message type and, via *NodeManager*, whether the sender of the *DATA* message exists (6.4, 6.5). If it exists, it receives a pointer to the corresponding *VirtualHost*.
6. The *PacketProxy* calls the *MsgHandler* of the *VirtualHost* to handle the packet (6.6).
7. The *MsgHandler* gets the *DATA* message and a new *RAWEtherFrame* is transmitted (6.7–6.10).
   - 7.1. The *DATA* message is read from the *Receive Buffer* (6.7).
   - 7.2. A new *RAWEtherFrame* packet is created (6.8).
   - 7.3. The original Ethernet frame of the *DATA* message is copied to the new packet.
   - 7.4. The destination address of the *RAWEtherFrame* packet is set to the destination MAC address of the original Ethernet frame.
   - 7.5. The packet is passed to the *WlanNIC* (6.9).
   - 7.6. The packet is transmitted inside the simulation model (6.10).
   - 7.7. The *RAWEtherFrame* message is transmitted to the corresponding *VirtualHost* (6.9, 6.10).

3.3.4. *Packet reception by an external node*

1. The *VirtualHost* receives a packet through the *WlanNIC* (6.11) and passes it to the *MsgHandler* (6.12).
2. The *MsgHandler* encapsulates the Ethernet frame inside a new *DATA* message, includes the infrastructural IP address and listening port of the corresponding external node and then transmits it to the *Interface Send Buffer* of the model (6.13, 6.14).
3. The *DATA* message is then forwarded to the external node (6.15).
4. The *PacketModeller* at the external node then decapsulates the packet and injects the Ethernet packet to the network stack of the node (via the virtual interface *vifx*) (4.7–4.11).

3.3.5. *Node configuration*

1. The wireless tools, such as the *iwconfig* utility, modify the wireless interface configuration of the external node.

2. The patched version of the *iwconfig* tool propagates all wireless configuration changes via *libvif* to the shared memory *vif* configuration (*vifd*).
3. Then, the *PacketModeller* is triggered to send a *CONFIGURATION* to the simulation model (4.3–4.6).
4. The simulation model processes the *CONFIGURATION* messages (6.1–6.7).
   - 4.1. The *PacketProxy* reads the *CONFIGURATION* message (6.4).
   - 4.2. It then checks whether a representation of the external node exists in the model with the help of the *NodeManager* (6.5).
   - 4.3. The *PacketProxy* then invokes the *MsgHandler* of the corresponding *VirtualHost* to process the *CONFIGURATION* message (6.6, 6.7).
   - 4.4. The *MsgHandler* stores the new wireless parameter values to the node database.
   - 4.5. Hence, the *VirtualHost* uses the new values for packet transmissions inside the model.

The nodes can modify the parameters of their wireless network interfaces by the wireless tools (*iwconfig* utility) at any time during the simulation. Therefore, even scenarios with dynamic multi-channel communication are possible, but simulation model has to support them.

4. Evaluation

VirtualMesh has to fulfil the following requirements in order to be useable for pre-deployment tests. VirtualMesh should not change the behaviour of network protocols and the application running on top of it. It has to be fully transparent for the upper layers. This has been verified in our functional evaluation in Section 4.3. In addition, neither bandwidth nor delays of the wireless network should be heavily influenced by VirtualMesh. Due to traffic interception, traffic redirection to a simulation model, and the optional node virtualization, the architecture of VirtualMesh introduces some additional delays to the system. We have performed several experiments in order to quantify these delays and detect possible bottlenecks. Furthermore, we quantify the effect of VirtualMesh on the network bandwidth.

4.1. Design alternatives

For node virtualization, either full virtualization or para-virtualization can be used. Full virtualization provides a complete simulation of the underlying hardware. A full-virtualized host uses the real device drivers, which then work on top of an emulated hardware layer. All software, including the operating system and device drivers, runs unmodified, in the same way as on the raw hardware. In contrast, para-virtualization introduces some adaptations to the guest operating system. The software interface of a para-virtualized machine is similar, but not identical, to that of real hardware. Therefore, the drivers for network and block devices are replaced. In our scenario, the para-virtualized host employs our standard embedded Linux system, which is also running on a real node. It makes use of the new para-virtualization feature of recent Linux kernels (*paravirt_ops*) that allows it to run on native hardware and as para-virtualized machine. The para-virtualized operating system kernel accesses the network and blocks devices through a XEN specific driver. The main advantage of para-virtualization is its improved performance compared to full virtualization. Moreover, the standard Linux kernel already includes the para-virtualization features by default. We have evaluated both approaches in Section 4.3.

4.2. Scenarios

To determine the round-trip times (RTTs), we have used simple ping (Internet control message protocol (ICMP) echo) measurements in a network with fixed node positions. The network consists of one computer hosting the simulation model (OMNeT++ 4.0) and one computer with the host virtualization (XEN 3.3.1), holding up to five virtual node instances. The two hosts (Pentium D 930 3 GHz, 2 GB random-access memory (RAM)) and the mesh nodes are interconnected by a 1 Gbps Ethernet network.

We evaluated the latencies/delays introduced by the service network, including virtualization and the traffic interception/redirection, and those introduced by the simulation model. In Figures 10–13, each data set represents measurement series of 1000 ICMP echoes with default sending rates of 1 packet/s. Figure 14 shows the resulting UDP throughput. The results are shown as boxplots, i.e. a bold line marks the median value, box lines represent the lower and upper quartiles and the whiskers mark the minimum and maximum values.

4.3. Results

For functional evaluation, we have tested several existing Linux applications, such as secure remote shell (ssh) and file transfers using the file transfer protocol (FTP) and secure copy (scp). These applications just work without any problems on top of the simulated network of VirtualMesh.

Firstly, we conducted tests to measure the network latencies between (a) two physical hosts, (b) a physical host and a para-virtualized host and (c) a physical host and a full-virtualized host. Our results in Figure 10 confirm that the additional delay introduced by

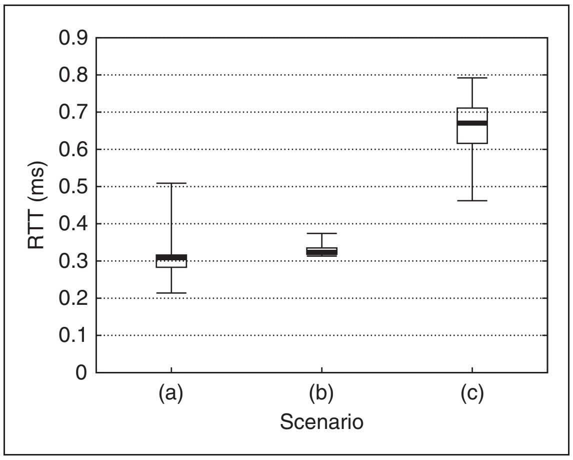


Figure 10. RTT between physical and (a) physical, (b) para-virtualized, (c) full-virtualized host.

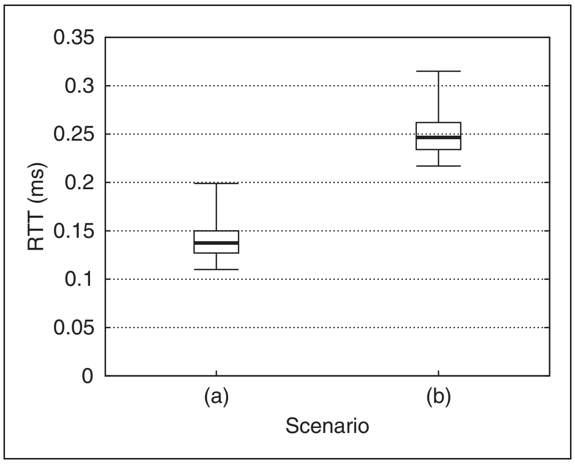


Figure 11. RTT between two virtualized nodes without *PacketModeller* (a) and with *PacketModeller* (b).

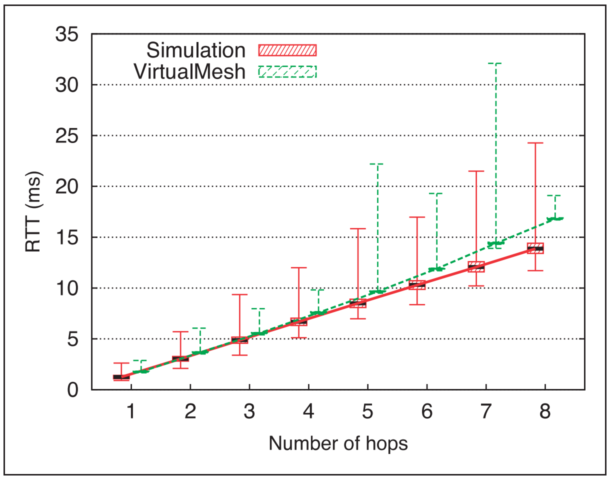


Figure 12. Comparison of RTT between pure simulation and VirtualMesh with an increased number of hops.

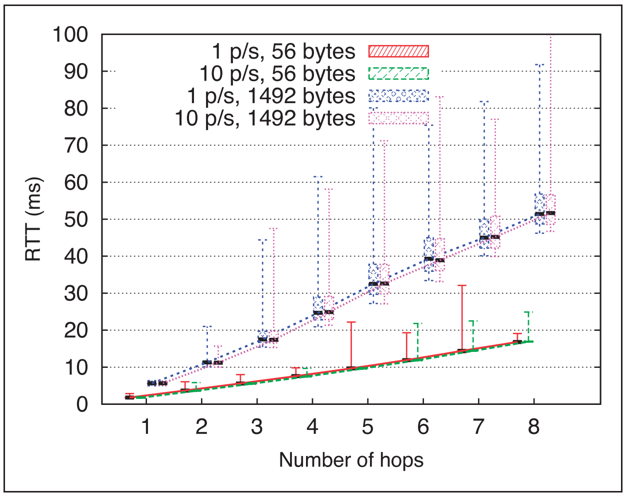


Figure 13. RTT for VirtualMesh with two sending rates and payload values.

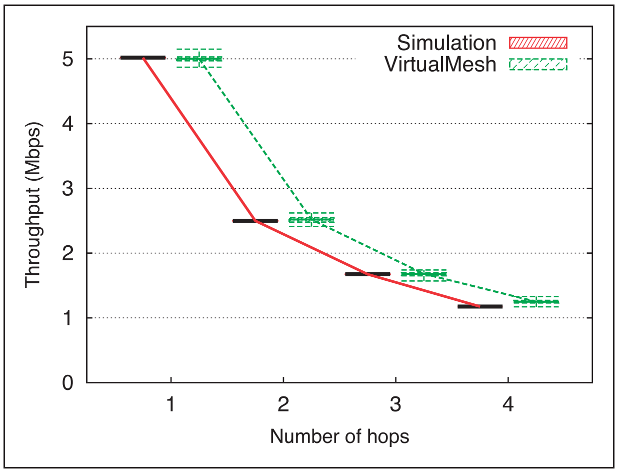


Figure 14. UDP throughput measurements using pure simulation and VirtualMesh.

virtualization is less than 0.2 ms for communication between a physical host and another physical host (a), or between a physical host and a para-virtualized host (b) is nearly negligible. If we replace the para-virtualized node with a fully virtualized node, the delay obviously significantly increases due to the hardware emulation layer (see (c) in Figure 10), although it still remains acceptable at 0.35 ms. Nevertheless, we decided to use para-virtualization for VirtualMesh, as it introduces nearly half the delay. The mandatory requirement of a low delay for traffic redirection can thus be confirmed by the tests. Moreover, the replacement of the Ethernet driver does not affect the accuracy of VirtualMesh.

The second experiment evaluates the delay introduced by the *PacketModeller* (see Figure 11). RTTs between two virtual hosts with and without traffic interception by the virtual interface *vifx* and the *PacketModeller* are measured. The result shows a moderate increase of less than 0.15 ms, which is absolutely negligible.

Our third experiment shows the additional delays introduced by packet redirection and packet processing through the simulation model compared to the pure simulation (see Figure 12). We use a chain topology with 2–9 nodes (1–8 hops) in the simulated network. The limitation to eight hops is reasonable, as paths are usually shorter in WMNs. In each of the eight scenarios, a series of ICMP echo packets (1 packet/s, 56 bytes payload) is sent from the first node to the last node in the row. Static routing is used in VirtualMesh and simulation. The pure simulation results show RTTs of approximately 1.6 ms per hop. In comparison, the RTTs measured per hop in the VirtualMesh scenario are, at 2 ms, slightly higher. The difference can be easily attributed to about 0.3 ms seen from the transmission in the infrastructural network (see Figure 10) and 0.1 ms due to two context switches between real space and simulated space. These results are significantly better than the results shown in our previous measurements.[6] We have considerably enhanced the performance of our VirtualMesh implementation by heavily restructuring the message flow inside the simulator and running the model in a 64-bit environment. Besides the direct comparison between simulation and VirtualMesh, we have also evaluated the behaviour of VirtualMesh when using a higher payload of 1492 bytes and an increased sending rate of 10 packets per second. In Figure 13, the results of VirtualMesh show the normal linear increase of RTTs with the hop count, independent of the payload.

Our fourth experiment compares the UDP bandwidth performance over multiple hops of VirtualMesh with that of a pure simulation (see Figure 14). The network is again set up in a chain topology with 2–5 nodes (1–4 hops). Inside the model, the raw data rate of the radio is set to 11 Mbps. As expected, the throughput values of VirtualMesh almost match the values of the pure simulation. The retrieved throughput values with 5 Mbps for the one-hop connection, 2.5 Mbps for the two-hop connection, 1.7 Mbps for the three-hop and 1.25 Mbps for the four-hop connection represent the normal significant decrease of the throughput, depending on the number of hops similar to the 1/(hop count) as shown in the literature.[26,27] The throughput values hence only depend on the used simulation model. VirtualMesh does not include any additional limitations concerning bandwidth in this setup. The higher variance seen in the results of VirtualMesh depends on the slightly different evaluation compared to the pure simulation.

From these results, we conclude that VirtualMesh is a valuable infrastructure for making pre-deployment tests of various communication protocols and applications in WMNs. VirtualMesh has shown no influence on throughput. It only introduces moderate additional delays due to traffic redirection and packet processing in the simulator. Therefore, it can be used for testing real software before going to the final deployment.

5. Conclusions

After development and evaluation with network simulators, wireless mesh communication solutions require extensive pre-deployment testing of their target platform implementations. This is difficult to achieve in a real test-bed, as irrepressible sources of interference exist. Furthermore, the variety of testing topologies is limited and mobility tests are impracticable. Therefore, we have designed VirtualMesh as a new testing architecture to be used before going to a real test-bed. VirtualMesh is based on the interception of wireless traffic at nodes and redirection to a simulation model that provides more flexibility and a controllable environment.

In comparison to other solutions, VirtualMesh provides a high integration of network emulation to real and virtualized hosts. The wireless drivers of the nodes are replaced by a virtual device that redirects traffic to an OMNeT++ simulation model instead of transmitting it over the air. This is fully transparent to the Linux network stack and the applications. The normal network stack and all applications can be used without any modification. Furthermore, even the standard configuration utilities can be used for wireless network configuration, as the virtual driver acts in the same way as a standard wireless network driver under Linux. All configuration parameters may be set using the usual configuration tools.

The focus of VirtualMesh is on the evaluation of wireless networks. It supports all common WMN scenarios, such as community networks and Internet sharing, with virtual nodes acting as gateways. Although VirtualMesh has been designed to test WMNs, it is not limited to them. The concept can also be applied to other wireless networks, e.g. mobile ad hoc networks (MANET).

A main advantage of VirtualMesh is the dynamic node management. Node registration and de-registration provides testing facilities even for management and software distribution frameworks that require rebooting of nodes, e.g. to update the operating system kernel or the communication software. During shutdown, a node just de-registers from the model and is not available until it registers again when being started.

We have evaluated the influence of VirtualMesh on delay and throughput. Our experiments have proven the full functionality of the VirtualMesh testing infrastructure. VirtualMesh introduces only negligible additional delays for traffic redirection and per real node inside a simulated path (less than 0.4 ms per hop). UDP throughput measurements show that the throughput only depends on the used simulation model. The results match the ones gained from pure simulation. VirtualMesh does not introduce additional limitations, making it a valuable tool for protocol developers and practitioners to test developed software for WMNs prior to actual deployment.

Similar to other emulation frameworks, one problem of VirtualMesh is that the simulation may be too slow and cannot keep pace with the injected network traffic. Thus, the simulation model of VirtualMesh needs to be run on a powerful machine and to communicate over a distinct and high-performance management network. However, there may still be an overload situation. Therefore, we plan to integrate the concept of synchronized network emulation[24] into VirtualMesh.

Even though VirtualMesh supports dynamic networks with nodes joining and leaving, as well as modifications of wireless parameters, some parameters, such as the node positions and mobility pattern, have to be pre-defined. This limitation can be removed by propagating position updates through configuration messages to the simulation model in the future. Other future enhancements are the support for channel sniffing, retrieval of signal-to-noise ratio (SNR) values and the full support of highly dynamic multi-channel scenarios.

VirtualMesh is available for download.[28]

Acknowledgements

We would like to thank Kirsten Dolfus, Paul Kim Goode and the anonymous reviewers for their valuable comments and careful reviews of the paper

Funding

The work presented in this article was partly supported by the Swiss National Science Foundation [grant number 200020-113677/1].

References

1. Akyildiz IF and Wang X. A survey on wireless mesh networks. *IEEE Comm Mag* 2005; 43(9): S23–S30.
2. Bicket JC, Aguayo D, Biswas S and Morris R. Architecture and evaluation of an unplanned 802.11b mesh network. In: *Proceedings of the 11th Annual International Conference on Mobile Computing and Networking (MOBICOM 2005)*, p.31–42, Cologne, Germany, August 28–September 2, 2005.
3. Draves R, Padhye J and Zill B. Routing in multi-radio, multi-hop wireless mesh networks. In: *Proceedings of the 10th annual international conference on Mobile computing and networking (MobiCom'04)*, p.114–128, New York, USA, 2004.
4. Karrer R, Sabharwal A, Knightly E. Enabling large-scale wireless broadband: The case for taps. In: *Proceedings of the 2nd Workshop on Hot Topics in Networks (Hot-Nets II)*, Cambridge, MA, USA, November 2003.
5. Engel M, Smith M, Hanemann S and Freisleben B. Wireless ad-hoc network emulation using microkernel-based virtual Linux systems. In: *Proceedings of the 5th EUROSIM Congress on Modeling and Simulation*, p.198–203, Cite Descartes, Marne la Vallee, France, September 6–10, 2004.
6. Staub T, Gantenbein R and Braun T. VirtualMesh: an emulation framework for wireless mesh networks in OMNeT++. In: *Proceedings of the 2nd International Workshop on OMNeT++ (OMNeT++ 2009)*, held in conjunction with the *2nd International Conference on Simulation Tools and Techniques (SIMUTools 2009)*, Rome, Italy, March 6–7, 2009.
7. Ivanov S, Herms A and Lukas G. Experimental validation of the ns-2 wireless model using simulation, emulation, and real network. In: *Proceedings of the 4th Workshop on Mobile Ad-Hoc Networks (WMAN'07)*, p.433–444, Bern, Switzerland, February 26–March 2, 2007.
8. Krop T, Bredel M, Hollick M and Steinmetz R. Jist/mobnet: combined simulation, emulation, and real-world testbed for ad hoc networks. In: *Proceedings of WinTECH'07*, p.27–34, New York, USA, 2007, ACM.
9. Zimmermann A, Gunes M, Wenig M, Meis U and Ritzerfeld J. How to study wireless mesh networks: A hybrid testbed approach. In: *Proceedings of Advanced Information Networking and Applications, 2007 (AINA'07)*, p.853–860, May 2007.
10. Zhang Y, Li W. An integrated environment for testing mobile ad-hoc networks. In: *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc'02)*, p.104–111, New York, USA, 2002, ACM.
11. Barham P, Dragovic B, Fraser K, Hand S and Harris T. Xen and the art of virtualization. In: *Proceedings of the 9th ACM symposium on Operating systems principles (SOSP)*, Bolton Landing, NY, USA, October 19–22, 2003.
12. Raychaudhuri D, Seskar I, Ott M, Ganu S, Ramachandran K, Kremo H, Siracusa R, Liu H and Singh M. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In: *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2005)*, volume 3, p.1664–1669, March 2005.
13. White B, Lepreau J, Stoller L, Ricci R, Guruprasad S, Newbold M, Hibler M, Barb C and Joglekar A. An integrated experimental environment for distributed systems and networks. In: *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, p.255–270, Boston, MA, USA, December 2002, USENIX Association.

14. Rizzo L. Dummynet: A simple approach to the evaluation of network protocols. *Comput Comm Rev* 1997; 27(1): 31–41.
15. White B, Lepreau J and Guruprasad S. Lowering the barrier to wireless and mobile experimentation. In: *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, USA, October 28–29, 2002.
16. Johnson D, Stack T, Fish R, Flickinger DM, Stoller L, Ricci R and Lepreau J. Mobile Emulab: a robotic wireless and sensor network testbed. In: *Proceedings of INFOCOM 2006. 25th IEEE International Conference on Computer Communications*. Barcelona, Spain, April 2006.
17. Beuran R, Nguyen LT, Latt KT, Nakata J and Shinoda Y. Qomet: A versatile wlan emulator. In: *Proceedings of the International Conference on Advanced Information Networking and Applications*, 0:348–353, 2007.
18. Beuran R, Nakata J, Okada T, Nguyen LT, Tan Y and Shinoda Y. A multi-purpose wireless network emulator: Qomet. In: *Proceedings of the International Conference on Advanced Information Networking and Applications Workshops*, 0:223–228, 2008.
19. Judd G, Steenkiste P. Repeatable and realistic wireless experimentation through physical emulation. In: *Proceedings of the 2nd Workshop on Hot Topics in Networks (Hot-Nets II)*, Boston, MA, USA, November 2003.
20. Borries K, Judd G, Stancil D and Steenkiste P. FPGA-based channel simulator for a wireless network emulator. In: *Proceedings of the IEEE 67th Vehicular Technology Conference (VTC2009-Spring)*, Barcelona, Catalunya, Spain, April 2009.
21. Bless R and Doll M. Integration of the freebsd tcp/ip-stack into the discrete event simulator omnet++. In: *Proceedings of the 36th conference on Winter simulation (WSC'04)*, p.1556–1561, 2004.
22. Jansen S, McGregor A. Performance, validation and testing with the network simulation cradle. In: *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation (MASCOTS'06)*, p.355–362, Washington, DC, USA, 2006.
23. Lacage M, Weigle M, Dowell C, Carneiro G, Riley G, Henderson T and Pelkey J. The Network Simulator ns-3′, http://www.nsnam.org/ (Access date: December 2009).
24. Weingärtner E, Schmidt F, Heer T and Wehrle K. Synchronized network emulation: Matching prototypes with complex simulations. *SIGMETRICS Perform Eval Rev* 2008; 36(2): 58–63.
25. Varga A. The omnet++ discrete event simulation system. In: *Proceedings of the European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic, June 6–9, 2001.
26. Sun D, Man H. Performance comparison of transport control protocols over mobile ad hoc networks. In: *Proceedings of the 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications,* 2: 83–87 vol. 2, Sep/Oct 2001.
27. Toh C-K, Delwar M and Allen D. Evaluating the communication performance of an ad hoc wireless network. *IEEE Trans Wireless Comm* 2002; 1(3): 402–414.
28. Staub T, Gantenbein R and Braun T. 'VirtualMesh', http://www.iam.unibe.ch/~rvs/research/software.html (Access date: June 2010).

Thomas Staub received his Master's Degree in Computer Science from the University of Bern in 2004. Since then he has been a researcher and PhD student in the Computer Networks and Distributed Systems research group at the University of Bern. His research interests include wireless mesh and sensor networks, in particular self-configuration, automatic deployment, multi-path routing and quality of service. He led a Swiss National Science Foundation project on WMNs and the work package on dissemination and training of the European FP6 IP project EuQoS. In 2009, he was in charge as project leader for the technology transfer project 'Wireless Mesh Networks for Interconnection of Remote Sites to Fixed Broadband Networks'.

Reto Gantenbein is a student in Computer Science at the University of Bern. Currently, he is doing his Master thesis in the Computer Networks and Distributed Systems research group. He has been working part time supporting the high-performance computing grid of the University of Bern during his studies.

Torsten Braun has been Professor of Computer Science of the University of Bern and head of the Computer Networks and Distributed Systems research group since 1998. Since 2007, he has been Director of the Institute of Computer Science and Applied Mathematics at the University of Bern. He has been a member of the board of trustees of Switch, which is responsible for the Swiss research and education network, since 2001.