Experiences with Node Virtualization for Scalable Network Emulation

Steffen Maier [*], Daniel Herrscher, Kurt Rothermel

University of Stuttgart, Institute of Parallel and Distributed Systems (IPVS)
Universitätsstr. 38, D-70569 Stuttgart, Germany
Phone: +49(711)7816–226, Fax: –424

Abstract

During the development of network protocols and distributed applications, their performance has to be analyzed in appropriate environments. Network emulation testbeds provide a synthetic, configurable network environment for comparative performance measurements of real implementations. Realistic scenarios have to consider hundreds of communicating nodes. Common network emulation approaches limit the number of nodes in a scenario to the number of computers in an emulation testbed. To overcome this limitation, we introduce a virtual node concept for network emulation. The key problem for node virtualization is a transparent, yet efficient separation of node resources. In this paper, we provide a brief survey of candidate node virtualization approaches to facilitate scalable network emulation. Based on the gathered insights, we propose a lightweight virtualization solution to achieve maximum scalability and discuss the main points regarding its implementation. We present extensive evaluations that show the scalability and transparency of our approach in both a traditional wired infrastructure-based, and in two wireless ad hoc network emulation scenarios. The measurements indicate that our solution can push the upper limit of emulation scenario sizes by a factor of 10 to 28. Given our emulation testbed consisting of 64 computers, this translates to possible scenario sizes of up to 1792 nodes. In addition to the evaluation of our virtualization approach, we discuss key concepts for controlling comprehensive emulation scenarios to support scalability of our system as a whole.

Key words: software performance evaluation, network emulation, mobile ad hoc network, scalability, virtual routing

[*] Corresponding author.
Email addresses: `maier@ipvs.uni-stuttgart.de` (Steffen Maier), `herrscher@ipvs.uni-stuttgart.de` (Daniel Herrscher), `rothermel@ipvs.uni-stuttgart.de` (Kurt Rothermel).

Published in Computer Communications, Volume 30, Issue 5, pages 943-956, March 2007
© Elsevier B.V. 2006.
http://dx.doi.org/10.1016/j.comcom.2006.08.018

1 Introduction

During the design and implementation of distributed applications and network protocols, it is essential to analyze the impact of various network environments on their performance. While mathematical analysis and simulations are commonly used in early design stages, measurements are used to confirm the theoretical results as soon as implementations become available. Such measurements usually compare the performance of one implementation in different network environments or of different implementations in the same network environment.

Comparative performance measurements in real environments are considered problematic for two reasons. First, especially in scenarios with mobile nodes and wireless networking, it is hard to obtain multiple comparable measurement runs. Secondly, resource requirements prohibit measurements in larger scenarios. Therefore, there is strong demand for synthetic network environments that can be parametrized in order to reproduce an original or fictitious network.

The process of introducing network properties that differ from the actual properties of the hardware in use is called *network emulation*. A *network emulation tool* is software capable of altering network traffic in a specified way. A facility consisting of a combination of flexible networking hardware and suitable emulation tools is called *network emulation testbed*. Network protocols and distributed applications subjected to performance measurements in a network emulation testbed are called *software under test*.

Comparative performance measurements for mobile computing scenarios, e.g. the evaluation of an ad hoc routing protocol, typically require large scenarios with hundreds of nodes. The analysis of new applications for traditional infrastructure-based networks, e.g. a large-scale location service, may also require a high number of nodes, since both the end systems and the intermediate systems of the underlying infrastructure have to be considered.

Common network emulation systems assume that one communicating node in an emulation scenario corresponds to one physical computer in an emulation testbed. This severely limits the scalability, since testbeds with the required number of hundreds of computers are typically not available.

However, a number of applications aiming at resource-poor devices, e.g. in mobile computing scenarios, only need a fraction of the resources that a testbed node can provide. Therefore, we propose to run several instances of the software under test on a single testbed node ("physical node," *pnode*) [1]. Each instance of the software under test has to be provided a separate execution environment ("virtual node," *vnode*). In this paper, we give a brief survey of candidate approaches for node virtualization. Based on these approaches, we present a transparent, yet lightweight and thus very scalable solution to node virtualization for network emulation testbeds. In addition to node virtualization in the system core, we also discuss key concepts that support scalability of our system as a whole. Our implementation supports scalable emulation not only of networks consisting of point to point links but also of shared media based networks such as mobile ad hoc networks and even arbitrary combinations for the emulation of hybrid networks.

The remainder of this paper is structured as follows. The Network Emulation Testbed, which we use as a basis for our scalable network emulation approach, is introduced in Section 2. In Section 3, we provide a brief survey of candidate node virtualization approaches. We choose one of the candidate approaches for our implementation, which we discuss in Section 4. In Section 5, we provide extensive measurements showing the scalability of our approach for two important kinds of scenarios: emulation of infrastructure-based networks and MANETs (mobile ad hoc networks). For the latter, we also present a comprehensive real life example for performance analysis of an ad hoc routing protocol. Furthermore, we discuss the achievable degree of transparency for the software under test. In Section 6, we address key concepts to fully support scalability of our system as a whole. We discuss related work in Section 7. Finally, we conclude the paper in Section 8.

2 Overview of the Network Emulation Testbed

The Network Emulation Testbed (NET) [2] at the University of Stuttgart provides the basis for our scalable network emulation approach. It consists of 64 PC-nodes connected by a monolithic, programmable gigabit switch, and a separate administration network for setup and control (see Fig. 1). Using IEEE 802.1Q VLAN (virtual LAN) technology, the gigabit switch is able to create an arbitrary connection topology between the nodes. Each point-to-point link or shared media network segment in an emulation scenario, e.g. a WLAN (wireless LAN) channel, is mapped to a uniquely tagged VLAN.

On a testbed node, several tagged VLANs represent several virtual network interfaces, each of which is assigned a separate instance of our custom emulation tool. Each tool introduces the desired artificial network properties. It enables the configuration of arbitrary bandwidth limitations, delays, and frame error loss ratios. Additionally, to enable the realistic emulation of shared media networks, the effects of a MAC (media access control) layer can be reproduced. At the present time, this tool is capable of emulating IEEE 802.3 (Ethernet) [3]. We are currently completing an extension of the tool to allow the emulation of the ad hoc mode of IEEE 802.11 WLAN.

Our network emulation tool provides the service level abstraction of an unreliable datagram service to the software under test (see Fig. 2). This is the lowest possible emulation abstraction feasible to be implemented in software. The tool is implemented as a virtual network device driver, and therefore completely transparent to implementations on the network layer. As a result, the protocol stack including the network layer and all higher layers can be considered as software under test.

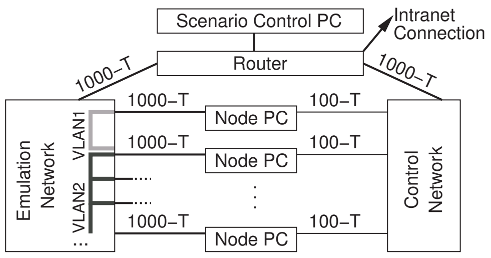

Fig. 1. The Network Emulation Testbed.

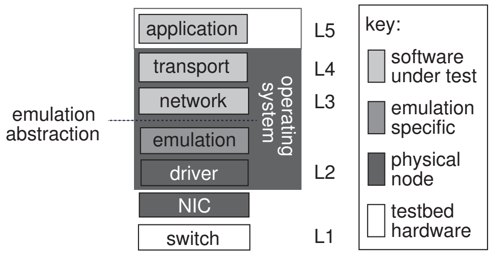

Fig. 2. Software under test and network emulation tools on a physical node in NET.

In order to control the distributed network emulation tools during an experiment, we pursue a hybrid architecture including a central scenario controller. The controller dynamically updates the parameters of the emulation tools. For MANET emulation, this includes changing connection quality and thus frame error rates between communicating nodes. The connection quality is automatically derived from the simulated node mobility and the application of different possible radio propagation models [4,5].

Without node virtualization, each node in an emulation scenario is mapped to a physical node of NET, which limits the scenario size to 64.

3 Approaches to Node Virtualization

In general, node virtualization provides a way to schedule formerly exclusive hardware resources to a number of consumers. With respect to network emulation, our consumers are execution environments for software under test, which is to be subjected to emulated network properties. We derive the following requirements from node virtualization and network emulation:

(1) Our paramount goal is scalability. This requires *minimal virtualization overhead* in order to preserve resources for the software under test.
(2) If two (or more) vnodes inside the same pnode communicate, their emulation tools should internally make use of *efficient intra-pnode communication* without affecting realistic emulation of network properties as perceived at the emulation abstraction by the software under test. This

requirement supports our paramount goal 1 by minimizing overhead due to the virtualization process.

(3) An execution environment introduced by node virtualization should be as *transparent* as possible for the software under test. This is important to support performance measurement of unmodified real implementations.

In the following, we present candidate node virtualization approaches and discuss their suitability regarding scalable network emulation. They all have in common that they allow multiple instances of software under test to be executed on top of the emulation abstraction interface shown in Fig. 2. For the discussion, we assume that the network stack is part of the kernel space, as is prevalent in commodity operating systems. Finally, we evaluate each approach for its suitability based on our requirements. The presented approaches can be classified into two main categories: virtual machines and virtual network stacks.

3.1 Virtual Machine

A straightforward way to introduce node virtualization is using a virtual machine (VM) approach. Instead of running an operating system (OS) directly on the bare hardware, a shim of software is inserted in between. This software – the virtual machine monitor (VMM) – schedules access of multiple guest operating systems to exclusive hardware resources managed by the VMM (Fig. 3).

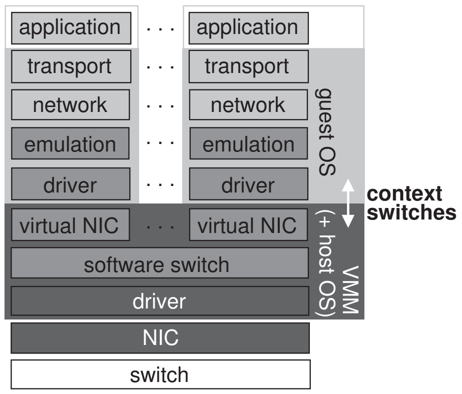

Fig. 3. Virtual machine approach.

In order to support emulated network parameters, we need to insert our emulation tool on top of network interface drivers inside each guest operating system. For communication with other vnodes on the same and other pnodes, a software switch forwards frames correspondingly.

3.1.1 Classical Virtual Machine

Classical virtual machines such as VMware Workstation [6] have in common that they support unmodified operating systems, and thus network stacks, in each guest instance. Therefore, they provide transparency with respect to software under test. However, context switches between guest OS and VMM happen whenever privileged commands are trapped. Since network communication causes such context switches, classical VMs imply considerable virtualization overhead limiting scalability. This is especially an issue for VMs, that do not virtualize a certain kind of system hardware, but e.g. the system call interface of the host OS, such as User-Mode Linux (UML) [7]. Even with a modified host OS such VMs only show comparable performance to e.g. VMware [8].

3.1.2 Lightweight Virtual Machine

Lightweight virtual machines such as VMware ESX [9], Denali [10] or Xen [11] directly access the host hardware without a host OS in order to reduce virtualization overhead. However, they may require custom or ported guest operating systems and are thus only partly transparent.

3.2 Virtual Network Stack

The virtual machine approach described in the previous subsection actually virtualizes more than is needed for network emulation. It would be sufficient to provide virtual execution environments for just the software under test, i.e. for exactly those layers above the emulation abstraction interface (Fig. 2). This can be accomplished with virtual network stacks (Fig. 4). In order to extend the virtualization of network and transport layer also to the application layer, sets of processes get associated with a certain network stack instance. Consequently, a vnode consists of the following sets: a set of processes on application layer, a set of sockets on transport layer, and a routing table on network layer.

In contrast to virtual machines, there is no more need for separate virtual network devices and their drivers. The emulation tool itself can appear as several instances of a virtual network device. In tight cooperation, a software switch forwards frames appropriately in order to allow communication between any vnodes. The virtualization overhead for virtual network stacks is as low as possible. Compared to virtual machine approaches, there are no redundant context switches and copy operations.

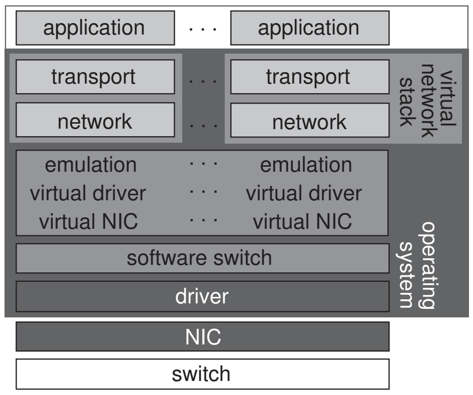

Fig. 4. Virtual network stack approach.

3.2.1 Duplicated Network Stack

Duplicated network stacks such as vimage [12] allow the flexible execution of different network stack implementations on the same pnode. However, they need extensive modifications to become fully virtualized and are thus hardly transparent.

3.2.2 Virtual Routing

Virtual routing [13] requires only the essential variables, that have to be allocated separately for each stack instance, to be touched. Thus, virtual routing is more transparent than duplicated network stacks. Though multiple instances are supported, only one specific implementation of a stack can be executed on a single pnode at a time. Yet, using different implementations on different pnodes remedies this partial flexibility.

3.3 Summary and Selected Approach

Tab. 1 shows a summary of the discussion in the previous subsections. We rate each approach on a scale of three levels with plus denoting fulfillment, a circle denoting partial fulfillment, and minus denoting restrictions with respect to our requirements.

For deriving the most suitable approach, we evaluate the fulfillment of our requirements in order of descending priority. While virtual machine approaches can be fully transparent and flexible, they do not fully comply with our paramount goal scalability. Virtual network stack approaches fulfill the requirement of low virtualization overhead and efficient intra-pnode communication. Of the two alternatives, virtual routing is more transparent. We thus consider virtual routing best suited for scalable network emulation.

Table 1
Comparison of candidate virtualization approaches.

virtualization approach	scalability	efficiency	transparency
3.1.1 classical virtual machine	−	−	+
3.1.2 lightweight virtual machine	○	○	○
3.2.1 duplicated network stack	+	+	−
3.2.2 virtual routing	+	+	○

4 Implementation

Virtual routing as discussed in the previous section fits our requirements best. Hence, we choose virtual routing along with a custom software switch that enables communication between any vnodes in an emulation scenario. Linux 2.4 serves as operating system for the implementation. The source code of our implementation is publicly available at our NET-project web page: *http://net.informatik.uni-stuttgart.de*. In the following, we describe the two main blocks of our approach traversing the layers from bottom to top.

4.1 Software Communication Switch

In the context of our network emulation testbed, each software switch introduces a "stacked" sub-switch using the emulation network connection as an uplink to the emulation switch (cp. to Fig. 1). A software switch resembles the functionality of a hardware Ethernet switch. It mediates both between vnodes located on the same pnode as well as between vnodes located on different pnodes. This provides transparent switching between any vnodes in a scenario.

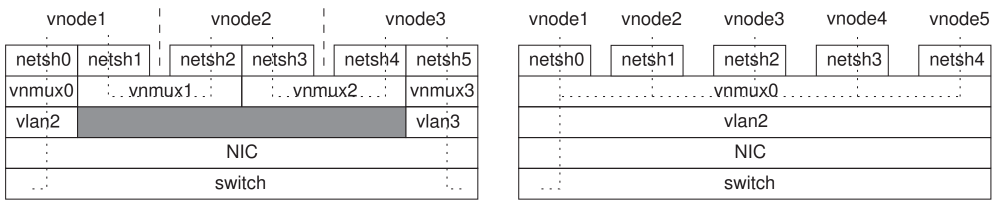

Fig. 5. Pnode configuration examples (arbitrary combinations are possible): link based (left), shared media based, e.g. MANET, (right).

In contrast to the software bridge already existing in Linux, we need one uplink to a real device and multiple local ends. Therefore we designed a custom Linux kernel module providing instances of a virtual switch network device "vnmux" (virtual node multiplexer) (Fig. 5). In order to get an uplink, this device can be internally bound to the driver of a real network device (NIC).

The latter could also be a tagged VLAN device which is in turn bound to a real network device. The bound device is put into promiscuous mode to be able to receive frames destined for local vnode devices. The emulation hardware switch takes care of filtering. It delivers only locally targeted frames after its learning phase, so that the software switch only has to process frames it really is responsible for. Processing frames is done without extra copying of payload data. This is essential to fulfill our requirement 2 for efficient intra-pnode communication. Switching decisions work with constant destination lookup time resulting in a complexity of $O(1)$. Our implementation supports transparent communication between any local and remote vnodes as well as unvirtualized pnodes. Additionally, link and shared media based pnode configurations (Fig. 5) can be mixed arbitrarily even on the same pnode. At the upper interface of the switch, virtual network devices provided by emulation tool instances ("netshX") register themselves to generate local switch "ports."

4.2 Virtual Routing

Virtual routing instances are located on top of our emulation tool's virtual network devices. Those virtual routing instances and applications on top of them represent possible software under test. We base our implementation on kernel patches for "Virtual Routing and Forwarding" (VRF) [14] version 0.100 by James R. Leu. VRF provides multiple instances of forwarding information bases (routing tables) as well as mechanisms to associate network devices, IPv4 UDP/TCP sockets, and processes with instances. User space tools exist for instance maintenance and for associating devices and processes with instances. Despite all these features, virtual routing is still not sufficiently transparent for application processes and common routing daemon implementations.

Therefore, we extend system interfaces that operate on routing tables – some IOCTLs and the protocol route-netlink – to work on the specific routing table of the VRF instance the calling process is associated with. Additionally, we extend the ip_queue feature of the protocol netlink-firewall to allow queueing of IPv4 packets to one process within each VRF instance. Thereby we gain full transparency for unmodified network applications including routing daemons.

To implement all of the above mentioned functionality, only limited modifications to a standard Linux 2.4.24 source tree are necessary. The modifications comprise 1409 lines of code, which consist of 416 additions, 980 changes, and 13 deletions.

5 Evaluation

In the following we provide an extensive evaluation of the building blocks as well as of the complete system. All measurements are performed on pnodes in our testbed equipped with an Intel Pentium 4 2.4 GHz processor and a Gigabit Ethernet adapter in a 32 bit, 33 MHz PCI bus. Passing through the different network stack layers from bottom to top, we start our evaluation with the software communication switch at the data link layer and show the accuracy of our network emulation tool in variably virtualized scenarios. Network and transport layer are treated twice since we consider two types of network requiring different routing algorithms: first a wired infrastructure based network, secondly a wireless ad-hoc network. For the wireless variant, we present one static scenario to draw comparisons with the wired network and one self-contained MANET scenario. The evaluation aims at showing the scalability of the system by comparing the non-virtualized cases to variably virtualized cases of the same scenarios. The experiments in Section 5.3 and 5.4 are conducted on Ethernet adapters without driver support for interrupt mitigation and are thus different from those presented in [1]. Concerning the software under test, we would like to point out that it is by no means limited to protocols on the network layer. After all, our load generators are processes on application layer communicating through sockets with the transport layer. Of course, more complex applications such as peer to peer systems can also be analyzed in our emulation environment without modification.

5.1 Software Communication Switch

Our software communication switch is a core component in our scalable emulation environment, since it has to switch frames quickly and at the lowest overhead possible. In order to show that it fulfills the expectations, we measure both the duration of switching decisions and the resulting throughput.

The scenario for measuring the duration of switching decisions consists of two pnodes connected by a point to point link. One of the pnodes hosts one switch instance with a varying number of vnode devices attached its local ports. For each measured value, the other pnode generates load by injecting 10^5 frames of sizes uniformly distributed between 64 and 1500 Bytes and randomly targeted at one of the software switch ports.

Fig. 6 shows the efficiency of unicast switching decisions. The average measured duration is about 98 ns, independent of the number of vnode devices per switch. The profiled machine code comprises 24 instructions, which take at least about 50 CPU clock cycles on the superscalar out-of-order core of our pnode CPU, if all data is available in the first level cache [15]. At the CPU frequency of 2.4 GHz, 50 clock cycles take about 20 ns. This marks a lower bound for the execution time. Taking cache misses into account, our measured average duration constitutes a reasonable value. A few spikes in the maxima up to 1388 ns are due to cache effects and appear rarely so that the average is close to the minimum of 88 ns. We conclude from these measurements that our implementation of the switching decision is highly efficient.

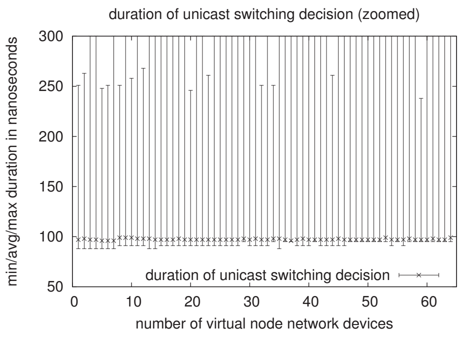

Fig. 6. Duration of unicast switching decision versus number of vnode devices.

The scenario for measuring switch throughput consists of one pnode with one switch instance having one uplink and a varying number of vnode devices attached. We vary frame sizes between 64 and 1500 Bytes. For each measured value we locally inject 10^6 frames of the same size. Fig. 7 shows constant throughput for unicast frames which only depends on the frame size. Small frames imply more overhead and thus less throughput. For comparison we measured a memory bandwidth of 1020 MByte/s with STREAM [16]. Obviously, frame handling overhead is the limiting factor in switch throughput. Nevertheless, a throughput of about 3 GBit/s can serve as an upper limit for aggregate link bandwidth inside one pnode and is 3 times larger than the external uplink over the Gigabit Ethernet network interface.

For broadcast frames their administration structure sk_buff (not the payload) has to be cloned on delivery for each local recipient. This is necessary since the receive path assumes exclusive administrative frame data structures. We also evaluate switch throughput for broadcast frames. The throughput for the starting value of 2 vnode network devices is slightly lower than for the unicast case because an additional frame clone has to be transmitted on the uplink (Fig. 8). With an increasing number of vnode devices per pnode, throughput decreases due to the overhead of cloning. Yet, aggregate switch throughput stays significantly above the memory bandwidth. However, in order to avoid any decrease, we plan to investigate possible improvements by sharing administration structures of broadcast frames between vnodes on the same pnode.

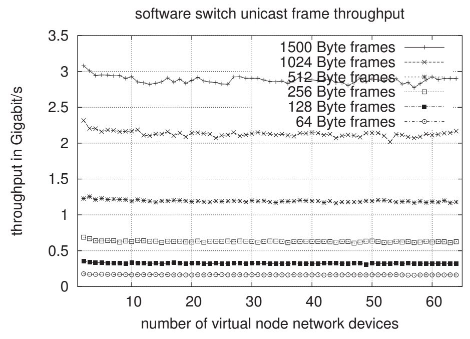

Fig. 7. Unicast frame throughput versus number of vnode devices.

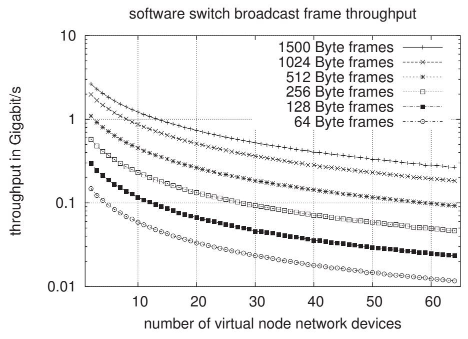

Fig. 8. Broadcast frame throughput versus number of vnode devices.

5.2 Network Emulation Tool

Our network emulation tool is able to accurately enforce specified network properties consisting of bandwidth limitation, delay, and frame loss ratio [2]. In this section, we show that our tool remains accurate in the virtualized case up to a machine dependent limit for the number of vnodes per pnode.

The scenario for measuring the accuracy of emulated network properties consists of a varying number of vnodes on a single pnode. n vnodes are interconnected in a chain of $n-1$ full duplex links having either limited bandwidth or specific delay in each direction (similar to Fig. 12). To measure loss ratio only one direction of each link is configured to lose frames. Each measured value is the result of one emulation run.

In order to measure the accuracy of bandwidth limitation, we use multiple instances of the tool *netio* to put load on each link by measuring maximum TCP throughput concurrently. Fig. 9 shows the results consisting of the measured average link bandwidth with minimum and maximum over all links, i.e. TCP flows. Depending on the number of vnodes, the specified bandwidth is accurately enforced by our network emulation tool. Up to an emulated bandwidth of 5 MBit/s, at least 64 vnodes can be hosted on a single pnode without loss of accuracy. 8 to 16 vnodes can be safely interconnected at 54 MBit/s and at least 4 vnodes can be hosted on a pnode in a Fast-Ethernet scenario with 100 MBit/s.

We measure ICMP round trip times (RTT) for 10^3 packets on each link concurrently to investigate the accuracy of delay emulation. Since the full duplex links are symmetric, the actual delay results from half the measured RTT. Fig. 10 shows mean, minimum and maximum RTT over all links. The results indicate that delay is emulated accurately independent of the number of vnodes per pnode. Thus, the emulation of delay scales perfectly with the degree of virtualization. The measured deviations from the average delay values stay within bounds of 5 ms and are due to the granularity of the timer used to introduce the delay [2].

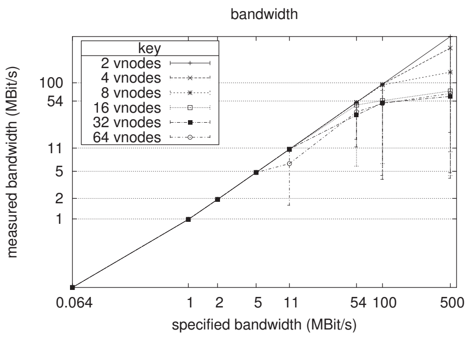

Fig. 9. Enforced versus specified bandwidth for different numbers of vnodes.

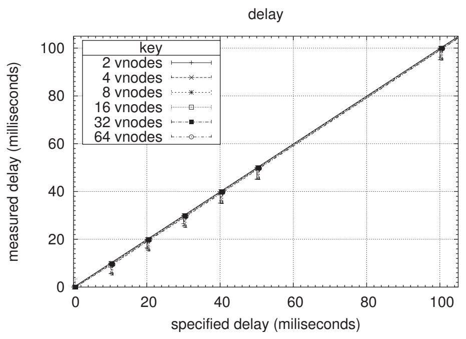

Fig. 10. Enforced versus specified delay for different numbers of vnodes.

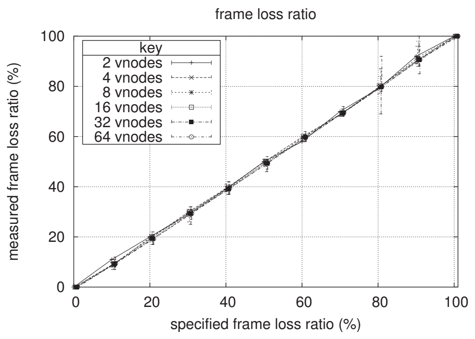

Fig. 11. Enforced versus specified loss ratio for different numbers of vnodes.

Fig. 11 depicts results showing the fidelity of emulated frame loss ratio. We put load on each link concurrently using adaptive ping with 10^3 packets. On average, frame loss emulation scales well with the number of vnodes per pnode. Minima and maxima – especially at a loss ratio of 80 or 90 % – are outliers and the deviation is below or equal to 2.7 for all measured values.

We conclude from our measurements that our network emulation tool is able to accurately enforce specified network properties over a wide range of virtualization degree.

5.3 Wired Infrastructure Based Network Emulation

Having treated the data link layer and the accuracy of our network emulation tool in the previous subsections, we now continue our evaluation of the network and transport layer in a wired infrastructure based network emulation scenario. The system model is described first. Afterwards, we present measurement results for the network and the transport layer. Additionally, we report on the system utilization caused by executing multiple vnodes on the same pnode.

The network topology of the scenario consists of a linear chain with a varying number of router nodes using static routing. Point to point links connecting the routers are full duplex and have an emulated limited bandwidth of 100 MBit/s in each direction. For the same scenario, we conduct the experiments twice differing only in the mapping of the scenario to the testbed hardware. First, we map each router to one real pnode to obtain reference values. Secondly, we place all routers inside vnodes on a single pnode except for the last router, which resides on a separate pnode without any virtualization (Fig. 12). Thereby we show that communication over the software switch works transparently, and mixing of arbitrarily configured pnodes is possible. Note that the layers of the real network stack implementation are always traversed on communication and forwarding even if the network traffic does not leave the left pnode except for the last hop.

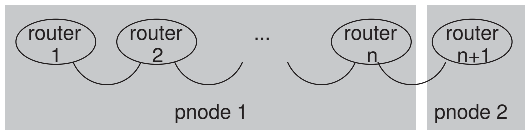

Fig. 12. Wired infrastructure emulation scenario, virtualized case.

On network layer, we measure ICMP RTT delays through the router chain. Each measured value consists of one experiment with 10^3 packets sent at a rate of 50 Hz. Fig. 13 shows linear increase of the mean ICMP RTT delay with an increasing number of hops in the routing chain. The left y-axis corresponds to the results of this infrastructure based scenario. The figure also contains measurement results for the wireless ad-hoc scenario discussed in the next section. For the variant with pnodes only, we had 51 of 64 nodes available at the time of the experiment. For the virtualized variant of the scenario, the slope is more flat than with pnodes only. This is because the software switch has lower communication delay than the hardware emulation switch. The emulation tool could compensate for that, if a particular scenario requires inter-node delays to be exactly the same. Delays occur within time bounds depicted in Fig. 14. Maxima occur only for the first packet until its destination is in the route cache. Thus, mean values from Fig. 13 and minima fall close together. Outliers in the maxima, such as with 40 vnodes, are neither an effect of network emulation nor of virtualization. It is normal behavior of the route cache implementation and happens indeterministically if the fastest code path cannot be executed. It also happens in the unvirtualized scenario variant even though not visible in the presented result set.

On transport layer, we measure TCP throughput over the router chain using the tool *iperf* with a TCP window size (socket send or receive buffer) of 512 KBytes and a maximum segment size (MSS) of 1448 Bytes (Fig. 15). The

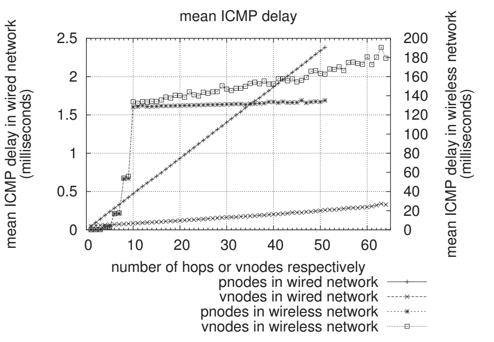

Fig. 13. Round trip time mean delay versus number of vnodes.

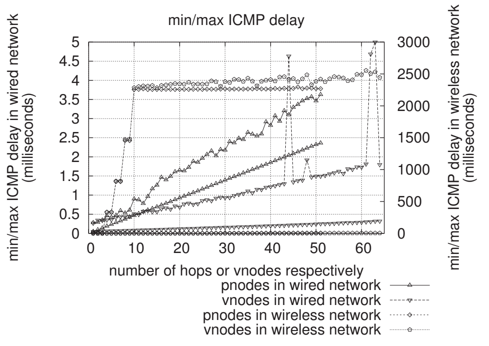

Fig. 14. Round trip time minima and maxima delay versus number of vnodes.

load generator is located on router 1 and the sink on router $n + 1$ (Fig. 12). It measures the throughput of one connection. In a subsequent measurement the location of generator and sink are swapped and another connection is measured. For the virtualized scenario, we observe different behavior in each direction. On the reverse direction (rx) from router $n + 1$ on pnode 2 to router 1 in a vnode on pnode 1, throughput starts dropping at 51 vnodes due to resource contention. On the forward direction (tx), throughput drops earlier at 12 vnodes. Both the sending TCP protocol with its timers in the leftmost vnode and all the other vnodes compete for the same resources of their shared pnode. TCP receive processing is no more the heavier side with modern CPUs [17]. Given the same resources, the TCP sender supports only lower throughput than the receiver could process.

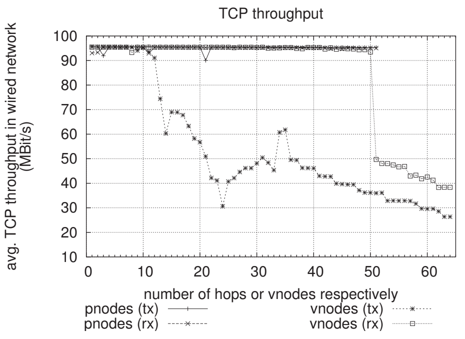

Fig. 15. Throughput versus number of vnodes.

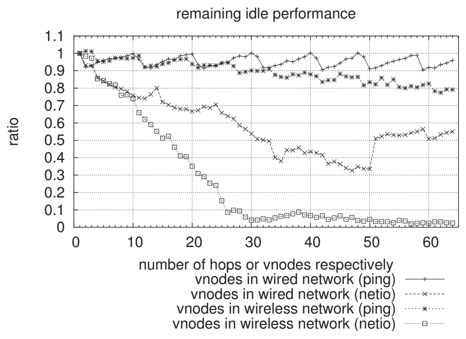

Fig. 16. Remaining system compute performance versus number of vnodes.

Measurements for throughput already showed deviation from the reality, if too many vnodes are hosted on the same pnode. In order to gain insight into system utilization, we measured the remaining idle performance on the pnode hosting vnodes while executing the two previously mentioned experiments. For this purpose, a custom tool consumes all available idle time quite aggressively while the load generating process of the previous two experiments is active. The tool reports on its compute progress which corresponds to the remaining idle compute time. The results of all measurement runs are normalized to the result for one vnode per pnode resembling the unvirtualized case. Fig. 16 shows only one plot per throughput measurement, since the TCP load generator measures both directions (rx and tx in Fig. 15) back-to-back. In general, the remaining idle performance decreases with an increasing number of vnodes per pnode. This is an indicator for possible resource contention due to virtualization.

Hosting too many vnodes per pnode leads to severe resource contention which can lead to measurement artifacts. Since we are interested in realistic results, the number of hosted vnodes is limited. For the above measurements, the earliest undesirable deviation from the unvirtualized case happens for TCP throughput at a number of 12 vnodes (Fig. 15). Given the scenario above and our testbed hardware with 64 pnodes, we thus can support scenario sizes of up to 704 nodes.

5.4 Wireless Ad hoc Network Emulation

Wireless ad hoc networks typically consist of a large number of communicating devices, which are often resource-poor. By using node virtualization, wireless ad hoc scenarios can be emulated with a meaningful number of devices on an affordable smaller number of computers in an emulation testbed. Hence, we evaluate the scalability of our approach for the emulation of such scenarios. As before, we describe the system model, followed by evaluation results for the network and transport layer as well as for system utilization.

Fig. 17 shows the emulation scenario. For comparison with the infrastructure scenario, we configured the virtual node positions and the emulated wireless network transmission range – depicted by dotted circles – such that the connectivity of the nodes resembles a chain. This is accomplished by a frame loss ratio for ingress traffic of zero for frames from reachable neighbors, and one for all others, as described in [4]. The wireless links between nodes are full duplex and have a limited bandwidth of 11 MBit/s. Here, we do not emulate the effects of a MAC layer, i.e. there are no frame collisions. Incorporating a MAC layer emulation as mentioned in Section 2 requires more resources and could reduce scalability. In this scenario, we use an implementation of the ad hoc on-demand distance vector routing protocol called AODV-UU [18] in version 0.8 as software under test. On each node an instance of the routing daemon is executed transparently with virtually no modifications necessary. Similar to the infrastructure scenario, we measure this scenario once with only pnodes and once with all vnodes on a single pnode, except for the last node, which resides on a separate pnode. The workload is the same as for the infrastructure scenario.

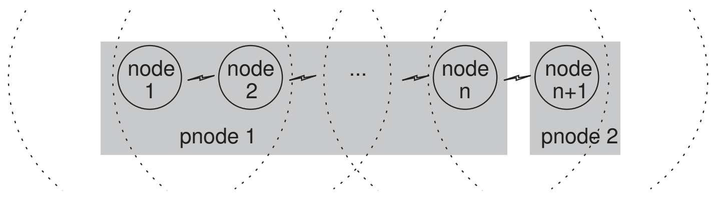

Fig. 17. Wireless ad hoc network emulation scenario, virtualized case.

On network layer, we measure ICMP RTT delays between node 1 and node $n + 1$ in the scenario (Fig. 17). The right y-axis in Fig. 13 corresponds to the measurement results for mean delay times. Starting with one hop we observe expanding ring search in combination with binary exponential backoff for outgoing route requests as described in [19], which is implemented by AODV-UU. Beyond the default time to live threshold, route requests work without expanding ring search leading to ICMP delays with linear increase starting at ten hops. For the virtualized variant, the slope is slightly larger for maxima and thus also mean RTTs due to increased latency on route establishment if multiple routing daemons compete for the resources of the same pnode. The right y-axis in Fig. 14 corresponds to minima and maxima in ICMP delay times. Maxima resemble multiples of the mean values due to route cache misses on route establishment. Minima show very flat linear increase being observed for established routes with route cache hits.

Measurement results for the transport layer are depicted in Fig. 18. TCP throughput starts deviating from the reference values in the unvirtualized scenario variant at about 29 hops. As a result of the lower limited bandwidth compared to the infrastructure scenario in the previous subsection, more virtual nodes can be executed on a pnode without interference.

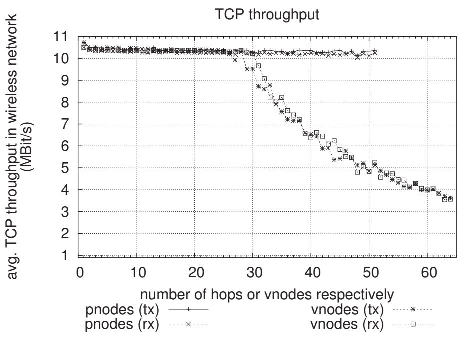

Fig. 18. Throughput versus number of vnodes.

Remaining idle performance for the virtualized wireless scenario is shown in Fig. 16. In addition to the forwarding operation on network layer as per the previous infrastructure based scenario, one ad hoc routing daemon is executed on application layer on each vnode. This results in higher system utilization, the more vnodes are hosted on a pnode. The consequence is remaining idle performance of about 10 % at a maximum of 28 vnodes.

The earliest undesirable deviation from the realistic reference values happens for TCP throughput at 29 vnodes per pnode (Fig. 18). Given the above scenario and our testbed hardware with 64 pnodes, we thus can support scenario sizes of up to 1792 nodes for similar wireless scenarios.

5.5 Comprehensive MANET Emulation

As a complement to the previous evaluations primarily investigating scalability, we show the utility of our system by means of an example for a scenario that is universally useful for developers to analyze the performance of a real protocol implementation. We follow [20] for the scenario model and evaluate AODV-UU 0.8 according to the metrics: packet delivery ratio and routing overhead. Each emulation run comprises 50 mobile nodes inside an area of $1500 \times 300\,\mathrm{m}^2$ and lasts 900 s. Each of 20 nodes starts uniformly distributed between 0 and 180 s generating constant bit rate load with 64 Byte packets at 4 Hz via UDP to one randomly chosen peer. Measurement results are averaged over two runs with different mobility trace. We compare the same implementation in two different network environments. First, for reference measurements the nodes move according to the random waypoint mobility model with a fixed transmission range of 250 m like in [20]. The second scenario aims to be more realistic and the nodes move along streets and choose destinations on them. The necessary geometric data corresponds to a part of the Stuttgart city center, for which we also precomputed realistic wave propagation taking static obstacles into account [5]. A central instance (see Section 6) controls the dynamic aspects of the scenario such as node mobility and resulting connectivity. We do not consider effects of the MAC layer (cp. to Section 5.4). To also evaluate the scalability of our emulation approach in such a real life scenario, we vary the total number of pnodes used to host the 50 vnodes.

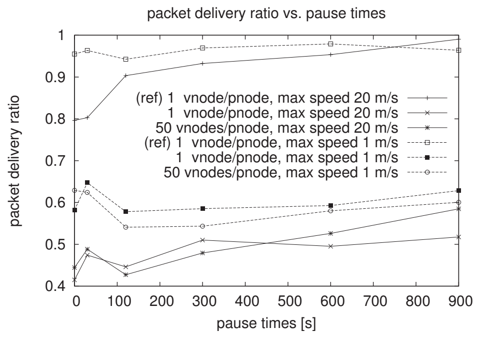

Fig. 19. Packet delivery ratio versus pause times.

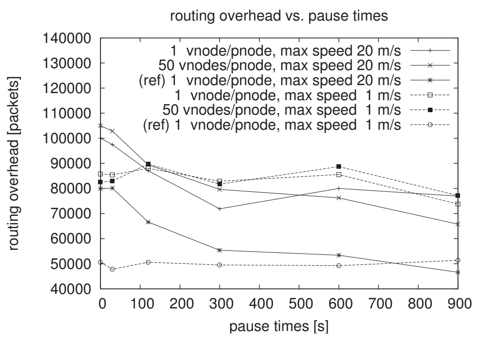

Fig. 20. Routing overhead versus pause times.

We observe reference values for the packet delivery ratio in Fig. 19 comparable to [20]. Using realistic wave propagation, the ratio is significantly lower because there is less connectivity due to obstacles. The reference values for routing overhead in Fig. 20 are different, since the AODV implementation in [20] does not use Hello messages for neighbor detection. We believe that this pays off for low speeds but causes more overhead at high speeds. Using realistic wave propagation, the overhead is a multiple of the reference because obstacles reduce connectivity. Low traffic load in the scenarios and sufficiently comparable results for different amounts of virtualization lead us to the conclusion that such a real life scenario can be emulated on just a single pnode. We expect realistic emulation of larger scenarios using a number of pnodes.

6 Scaling Emulation Scenario Control

In the previous section, we have primarily evaluated the scalability of node virtualization as system core for scalable network emulation. We now discuss key concepts for controlling comprehensive emulation scenarios to fully support scalability of our system as a whole.

In our system, each communicating node specified in a network emulation scenario model is represented by a vnode in the testbed. In order to build, execute, and control a complete scenario consisting of a potentially large number of vnodes, we provide a central scenario controller (see Section 2). The scenario controller uses communication on the administration network for two types of controlling purposes. First, it executes commands remotely on the pnodes for scenario setup. Secondly, dynamic parameter updates are sent to the distributed emulation tools. The controller does *not* process any network traffic but only controls the tools, that actually process the traffic in a distributed fashion. Since direct control of all vnodes in the scenario does not scale, we use a hierarchical approach. For this, we introduce two proxies on each pnode. The controller only communicates with those proxies. One proxy takes care of remote command execution. The other proxy demultiplexes update messages for emulation tool instances on a pnode. Additionally, we use a reliable transport protocol (TCP in this case) for update messages, since unreliable datagram messages would be dropped for scenario sizes of 1000 or more vnodes due to missing flow control.

For MANET scenarios, the controller can provide a graphical user interface (GUI), that visualizes geographic node positions. The user may select a node to continuously display its current radio propagation map as a potential sender [5] along with its neighbors as depicted in Fig. 21. This helps the user to get a valuable impression of the causality between node movement and dynamic change of network topology at a glance. The update frequency of the GUI is entirely decoupled from the frequency of update messages sent to the emulation tools. In order to ensure that the control task can keep up with real time, we assign it a higher priority.

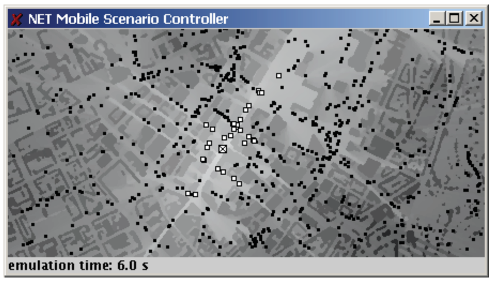

Fig. 21. Visualization of a MANET emulation experiment ($2225 \times 1065\,\mathrm{m}^2$).

7 Related Work

In this section, we review existing scalable network emulation approaches. All approaches have in common that they place components of the software under test to certain, different positions within an emulated network scenario. We classify the architectures to build the emulated network in centralized and distributed.

7.1 Centralized Network Emulation

Centralized approaches emulate a whole scenario within a single instance of a network emulation tool. The traffic that can be handled by the central instance constitutes the upper limit for the scalability of these approaches.

Ns-e [21] is an emulation extension of the well-known network simulator ns-2 [22]. The scalability of ns-e depends on the amount of traffic in the scenario. For a typical MANET experiment, a scenario size of about 50 nodes is possible [23]. To some extent, this can be alleviated by extending the discrete event simulation into a parallel engine [24].

ModelNet [25] is a parallel network emulator. It is primarily designed to emulate a given network topology of point-to-point links. The topology is partitioned among a cluster of emulation computers. Each cluster node processes network packets through internal arbitrarily connected links and routing instances. Computers running the software under test have to be externally connected to the central emulation cluster. Several instances of the software under test can run on each such edge node, which makes the approach scalable. However, the interface to the emulated network is based on socket calls, which restricts the software under test to the application layer. Existing implementations of network protocols cannot be analyzed but have to be specifically re-implemented for the cluster nodes. This is also true with MobiNet [26], which is an extension to emulate MANET scenarios. The presented 802.11 MAC emulation in MobiNet is completely centralized and only works on a single core node. For a MANET scenario the authors report scalability up to 200 emulated nodes with a re-implemented MANET routing protocol.

vBET [27] is an approach designed for emulating a network scenario on a single computer. It makes use of User Mode Linux (UML) [7] in order to provide virtual machines as execution environment for multiple virtual nodes on one computer. In combination with additional network emulation tools, it is possible to connect software under test to an emulated scenario. Connecting multiple of such vBET computers could allow larger scenarios. However, the use of UML's virtual machine concept introduces considerable overhead and thus limits the number of virtual nodes per computer. The authors report a maximum throughput for their software switch between vnodes of 128 MBit/s, which is more than an order of magnitude below our approach. vBET is more suitable for qualitative analysis than comparative performance analysis.

7.2 Distributed Network Emulation

Distributed approaches connect several instances of a network emulator together to form a comprehensive scenario.

Empower [28] allows the emulation of multiple routing instances on one computer, making up a link-based or wireless network topology. Each connection to the emulated network is mapped to a physical link of an existing hardware network interface. The authors equip each testbed node with several network cards to increase scalability. The number of network interfaces per pnode limits the number to a few vnodes per pnode.

Entrapid [29] virtualizes the network stack in user space and thus provides multiple execution environments for software under test on a single computer. In combination with network emulation tools connecting such virtualized stacks, the emulation of network scenarios is possible. However, the software under test has to be adapted in order to interact with the user space network stacks. The packet processing in user space also introduces considerable timing inaccuracies, compared to real network stacks. Thus, such approaches are more suitable for testing than performance evaluation.

Vimage [12] virtualizes the network stack in the kernel. While common operating systems support one single instance of a network stack, vimage supports multiple independent instances. To accomplish this, the stack is modified to have all formerly global instance variables independently available for each stack instance. Processes are associated with a certain network stack instance.

Thus, the virtualization is transparent for software under test on the application layer. In combination with the network emulation tool dummynet [30], it is possible to emulate link-based scenarios in a scalable way. However, modifying all instance variables and access to them incorporates substantial changes to the network stack and thus the software under test. In [12] the authors report TCP throughput of 420 MBytes/s over 15 routing hops on a single machine with a slightly faster processor than used in our evaluation. Though scaling significantly better than a VMware based virtualization approach, the throughput was measured in a best case without any introduction of emulated network properties such as bandwidth limitation or delay and is thus hardly comparable to our results. Emulated network properties are however essential for network emulation and imply emulation overhead due to timer management reducing the accumulated throughput that can be realistically emulated.

The emulation testbed Netbed [31] supports scalable network emulation by introducing virtual nodes [32] on the basis of BSD jails [33] and multiple routing tables. Netbed supports the emulation of scenarios with wired links. While it is possible to link real wireless nodes to an emulated scenario [34], there is no support for the reproducible *emulation* of wireless networks.

8 Summary and Conclusion

Network emulation testbeds provide a synthetic, configurable network environment for comparative performance measurements of distributed applications and protocols. Common approaches limit the scenario size to the number of computers in the testbed, whereas meaningful emulation scenarios often require hundreds of communicating nodes. Testbeds of such sizes are hardly available.

In this paper, we propose to execute multiple instances of the software under test on a single testbed computer. Therefore, we introduced virtual nodes providing the software under test with a virtual execution environment with respect to the network stack. From a set of candidate node virtualization approaches, we chose the most lightweight approach fulfilling our paramount goal for scalability. In addition to our emulation software tools, we implemented an efficient software communication switch, extensions to “Virtual Routing and Forwarding” for Linux by James R. Leu, and a hierarchical scenario control mechanism. We showed the utility of the implemented network emulation system with a comprehensive scenario and provided an extensive evaluation of the system. For a wired infrastructure-based and a wireless ad hoc network emulation scenario, our measurement results show that node virtualization can increase the possible scenario size by a factor of 10 or 28, respectively.

Given our testbed hardware with 64 physical emulation nodes, this translates to scenario sizes of up to 1792 nodes.

Clearly, for scenario sizes of several hundred nodes, it is no longer possible for an experimenter to manually map the nodes from an emulation scenario to the available testbed computers. Thus, our next step is an automated mapping based on constraints that evolve from the requirements of a scenario description and offerings of the testbed hardware. While we investigated the possible degree of virtualization by comparing measurements to the non-virtualized variant of a scenario, this procedure is not always desired or even possible. Therefore, we are introducing quality criteria for realistic network emulation which can be monitored in a lightweight fashion in situ while executing an experiment. In case of undesired resource contention due to virtualization, the experimenter will be informed and may decide to modify the mapping of vnodes to pnodes in order to prevent contention in another emulation run.

Acknowledgements

This work is partially funded by the Deutsche Forschungsgemeinschaft (German Research Foundation) under grant DFG-GZ RO 1086/9-1. We thank Marcus Handte and the anonymous reviewers for their valuable comments on the paper.

References

[1] S. Maier, D. Herrscher, K. Rothermel, On Node Virtualization for Scalable Network Emulation, in: Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005), Philadelphia, PA, USA, 2005, pp. 917–928.

[2] D. Herrscher, K. Rothermel, A Dynamic Network Scenario Emulation Tool, in: Proceedings of the 11th International Conference on Computer Communications and Networks (ICCCN 2002), Miami, FL, 2002, pp. 262–267.

[3] D. Herrscher, S. Maier, K. Rothermel, Distributed Emulation of Shared Media Networks, in: Proceedings of the 2003 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2003), Montréal, Quebec, Canada, 2003, pp. 226–233.

[4] D. Herrscher, S. Maier, J. Tian, K. Rothermel, A Novel Approach to Evaluating Implementations of Location-Based Software, in: Proceedings of the 2004 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2004), San Jose, CA, 2004, pp. 484–490.

[5] I. Stepanov, D. Herrscher, K. Rothermel, On the Impact of Radio Propagation Models on MANET Simulation Results, in: Proceedings of the Seventh IFIP International Conference on Mobile and Wireless Communication Networks (MWCN 2005), Marrakech, Morocco, 2005.

[6] J. Sugerman, G. Venkitachalam, B.-H. Lim, Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, in: Proceedings of the 2001 USENIX Annual Technical Conference, Boston, MA, 2001, pp. 1–14.

[7] J. Dike, A user-mode port of the Linux kernel, in: Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta, GA, USA, 2000.

[8] S. T. King, G. W. Dunlap, P. M. Chen, Operating System Support for Virtual Machines, in: Proceedings of the 2003 USENIX Annual Technical Conference, San Antonio, TX, USA, 2003, pp. 71–84.

[9] C. A. Waldspurger, Memory Resource Management in VMware ESX Server, in: Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02), Boston, MA, USA, 2002, pp. 181–194.

[10] A. Whitaker, M. Shaw, S. D. Gribble, Scale and Performance in the Denali Isolation Kernel, in: Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02), Boston, MA, USA, 2002.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the Art of Virtualization, in: Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing, NY, USA, 2003, pp. 164–177.

[12] M. Zec, M. Mikuc, Operating System Support for Integrated Network Emulation in IMUNES, in: Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (2004 OASIS), Boston, MA, USA, 2004, pp. 3–12.

[13] K. Kourai, T. Hirotsu, K. Sato, O. Akashi, K. Fukuda, T. Sugawara, S. Chiba, Secure and Manageable Virtual Private Networks for End-users, in: Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks (LCN'03), Bonn/Königswinter, Germany, 2003, pp. 385–394.

[14] J. R. Leu, Linux virtual routing and forwarding, http://linux-vrf.sourceforge.net (2004).

[15] Intel, USA, IA-32 Intel Architecture Optimization Reference Manual, order number: 248966-011 (2004).

[16] J. D. McCalpin, Memory Bandwidth and Machine Balance in Current High Performance Computers, IEEE Technical Committee on Computer Architecture (TCCA) Newsletter (1995) 19–25.

[17] A. P. Foong, T. R. Huff, H. H. Hum, J. P. Patwardhan, G. J. Regnier, TCP Performance Re-Visited, in: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2003), Austin, TX, USA, 2003, pp. 70–79.

[18] H. Lundgren, E. Nordström, C. Tschudin, Coping with Communication Gray Zones in IEEE 802.11b based Ad hoc Networks, in: Proceedings of the 5th ACM International Workshop on Wireless Mobile Multimedia (WoWMoM'02), Atlanta, GA, USA, 2002, pp. 49–55.

[19] C. E. Perkins, E. M. Royer, Ad-hoc On-Demand Distance Vector Routing, in: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications (WMCSA'99), New Orleans, LA, USA, 1999, pp. 90–100.

[20] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, J. Jetcheva, A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols, in: Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '98), Dallas, TX, USA, 1998, pp. 85–97.

[21] K. Fall, Network Emulation in the Vint/NS Simulator, in: Proceedings of the Fourth IEEE Symposium on Computers and Communications (ISCC'99), Red Sea, Egypt, 1999, pp. 244–250.

[22] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, H. Yu, Advances in Network Simulation, IEEE Computer 33 (5) (2000) 59–67.

[23] Q. Ke, D. A. Maltz, D. B. Johnson, Emulation of Multi-Hop Wireless Ad Hoc Networks, in: Proceedings of the Seventh International Workshop on Mobile Multimedia Communications (MoMuC 2000), IEEE Communications Society, Tokyo, Japan, 2000.

[24] G. F. Riley, R. M. Fujimoto, M. H. Ammar, A Generic Framework for Parallelization of Network Simulations, in: Proceedings of the 7th International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'99), College Park, MD, USA, 1999, pp. 128–135.

[25] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, D. Becker, Scalability and Accuracy in a Large-Scale Network Emulator, in: Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI '02), Boston, MA, USA, 2002.

[26] P. Mahadevan, A. Rodriguez, D. Becker, A. Vahdat, MobiNet: A Scalable Emulation Infrastructure for Ad Hoc and Wireless Networks, Technical Report CS2004-0792, Department of Computer Science, University of California, San Diego (Jun. 14 2004).

[27] X. Jiang, D. Xu, vBET: a VM-Based Emulation Testbed, in: Proceedings of the ACM SIGCOMM 2003 Workshops, Karlsruhe, Germany, 2003, pp. 95–104.

[28] P. Zheng, L. M. Ni, EMPOWER: A Network Emulator for Wireless and Wired Networks, in: Proceedings of the Conference on Computer Communications (INFOCOM 2003), Vol. 3, San Francisco, CA, USA, 2003, pp. 1933–1942.

[29] X. W. Huang, R. Sharma, S. Keshav, The ENTRAPID Protocol Development Environment, in: Proceedings of the Conference on Computer Communications (INFOCOM '99), Vol. 3, New York, NY, USA, 1999, pp. 1107–1115.

[30] L. Rizzo, Dummynet: a simple approach to the evaluation of network protocols, ACM Computer Communication Review 27 (1) (1997) 31–41.

[31] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, A. Joglekar, An Integrated Experimental Environment for Distributed Systems and Networks, in: Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02), Boston, MA, USA, 2002, pp. 255–270.

[32] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, J. Lepreau, Feedback-directed Virtualization Techniques for Scalable Network Experimentation, Flux Group Technical Note FTN-2004-02, School of Computing, University of Utah (May 2004).

[33] P.-H. Kamp, R. N. M. Watson, Jails: Confining the omnipotent root, in: Proceedings of the 2nd International System Administration and Networking Conference (SANE 2000), Maastricht, The Netherlands, 2000.

[34] B. White, J. Lepreau, S. Guruprasad, Lowering the Barrier to Wireless and Mobile Experimentation, in: Proceedings of the First Workshop on Hot Topics in Networks (Hotnets-I), Princeton, NJ, USA, 2002.

Steffen Maier is a scientific staff member in the Distributed Systems research group at the University of Stuttgart, Germany. His research interests include scalability of network emulation. Maier holds a Dipl.-Inf. (MS) degree in computer science from the University of Stuttgart.

Daniel Herrscher is a scientific staff member in the Distributed Systems research group at the University of Stuttgart. His research interests include realistic network emulation of mobile communication. Herrscher holds a Dipl.-Inf. (MS) degree in computer science from the University of Stuttgart.

Kurt Rothermel is a professor in the Distributed Systems research group at the University of Stuttgart. His research interests include performance evaluation of distributed systems, context aware and adaptive systems, and sensor networks. Rothermel received a PhD in computer science from the University of Stuttgart. He is a member of the ACM and the GI.