Time Jails: A Hybrid Approach to Scalable Network Emulation

Andreas Grau[†], Steffen Maier, Klaus Herrmann, Kurt Rothermel
Universität Stuttgart
Institute of Parallel and Distributed Systems (IPVS)
Universitätsstr. 38, D-70569 Stuttgart, Germany
E-mail: {grau,maier,herrmann,rothermel}@ipvs.uni-stuttgart.de

Abstract

It is essential to evaluate the performance of newly developed distributed software and network protocols. Network emulation enables reproducible evaluation of unmodified real implementations. Software built for distributed systems, such as a large scale peer-to-peer system, requires evaluation scenarios with thousands of communicating nodes. Two approaches for scaling network emulation to such scenario sizes have been proposed in the literature: node virtualization and time virtualization. Node virtualization allows maximizing the utilization of standard hardware used for emulation experiments. Time virtualization enables trading experiment duration for virtually increased resources of the hardware. It stands to reason that a combination of those two approaches may increase scalability even further. However, in existing combinations, either node virtualization implies relatively high overhead or time virtualization requires modifications of the test subject implementation.

In this paper, we present a novel hybrid approach called Time Virtualized Emulation Environment (TVEE). It integrates node virtualization with low overhead and time virtualization, which is transparent to the execution of test subjects. We introduce virtual time based on epochs to enable better dynamic hardware utilization during long lasting experiments. Additionally, a mechanism similar to soft timers ensures an accurate reproduction of network properties in the time virtualized emulation. Our evaluations show the accuracy and scalability of time virtualized network emulation. Comparing TCP throughput, TVEE outperforms other approaches using an event based virtual time by an order of magnitude.

[†]Funded by the Deutsche Forschungsgemeinschaft (German Research Foundation) under grant DFG-GZ RO 1086/9-2.

1 Introduction

The development of new distributed applications and communication protocols requires mechanisms to evaluate the performance of the corresponding implementations. Well known approaches for performance evaluation are simulation and live testing. However, simulators usually need a special implementation of a simulation model and cannot be used to evaluate unmodified real prototypes. Live testing requires an appropriate hardware environment. Setting up such an environment can be prohibitively expensive. Especially in wireless environments, measurement results are also not reproducible.

Network emulation solves those limitations by supporting the execution of unmodified real implementations within a controlled synthetic environment. We consider all layers above data link layer, including network, transport, and application layer, as software under test, which is referred to as test subject. On inexpensive commodity hardware, an emulation tool implemented in software reproduces specified network properties. The reproduction is transparent to the execution of the test subjects on the same computer. Multiple such computers interconnected by a flexible networking hardware allow the evaluation of scenarios with as many communicating nodes as computers are available.

Realistic evaluation scenarios often require thousands of nodes. In order to scale network emulation to such scenario sizes, different solutions have been proposed in the literature. We classify them into parallelization, abstraction, node virtualization, and time virtualization. Parallelization is typically used to take advantage of the distributed execution of simulation based approaches [15, 16, 4]. When executing real implementations of test subjects possible rollbacks of optimistic approaches are very expensive, which leads to use of conservative approaches. However, depending on the delays of the simulated network, these approaches require a high synchronization effort. Other approaches increase scalability by introducing abstractions to

Published in Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation, (PADS'08), pages 7-14, Roma, June 2008.
© IEEE 2008
http://dx.doi.org/10.1109/PADS.2008.19

the emulation model, e.g. flow-based instead of packet-based operation [10]. However, flow-based emulation prohibits the use of real network and transport layer implementations and, therefore, can only be used to evaluate real application layer implementations. Node virtualization approaches partition physical emulation computers (pnodes) into multiple virtual nodes (vnodes) and improve hardware utilization [8, 1]. Since they usually execute the test subjects in real-time, scalability is limited by the available hardware resources. Time Virtualization provides the possibility to virtually increase resources of an emulation computer by executing the test subject slower than in real-time [7]. Existing approaches employ the concept of virtual machines to provide virtual time transparently to the unmodified test subjects and to provide node virtualization at the same time. Yet, such node virtualization limits scalability by the relatively high implied virtualization overhead.

In this paper, we present three main contributions: (1) a hybrid node virtualization approach having low overhead and supporting time virtualization transparent to the test subjects, (2) virtual time based on epochs for maximizing hardware utilization during long lasting experiments with minimum synchronization overhead for distributed network emulation, and (3) a mechanism to ensure the accuracy of frame delays in a network emulation tool despite time virtualization.

The remainder of this paper is structured as follows. In Section 2, we discuss existing mechanisms for improving scalable network emulation with focus on node virtualization and time virtualization. We propose the most promising combination of the two mechanisms and the concept of time epochs in Section 3, where we also describe how to integrate the emulation of network properties and improve its accuracy. In Section 4, we evaluate the scalability of our prototype before we conclude the paper in Section 5.

2 Classification of Existing Mechanisms and Related Work

From our main objective of scalable network emulation, we derive the following two requirements. Emulation should imply minimum overhead, in order to leave as much hardware resources as possible to the execution of a large number of test subjects. We are especially interested in low overhead of computation, network input/output, and main memory. Emulation should be as transparent as possible for the execution of unmodified test subjects. With respect to node virtualization, this implies resource partitioning for different classes of operating system objects: processes, filesystems, and the protocol stack consisting of sockets, routing tables, and network devices. In the following, we discuss the fulfillment of those requirements for different approaches and related work in the area of scalable network emulation. We classify them into parallelization and abstraction, node virtualization, and time virtualization.

2.1 Parallelization and Abstraction

Simulating networks requires large computation power. In order to emulate networks, simulators need to be extended by a facility to process real network traffic and by a real-time scheduler which amplifies this demand. In the following, we discuss approaches to reduce computation load on simulation computers.

The concept of parallelization reduces load on single computers by distributing a simulator to multiple computers [15]. Each computer simulates a fraction of the network.

IP-TNE [16] uses a distributed event simulator combined with a real-time traffic capturing component that enables the evaluation of real applications within a simulated network. In contrast to our approach, no node virtualization is used to maximize hardware usage and emulation runs at real-time. Using a conservative strategy for distributed event simulation, the synchronization overhead is higher than using epoch based virtual time.

Whereas approaches based on parallelization distribute simulation efforts to multiple computers, other approaches reduce the load on computers by introducing abstractions in the emulation model, e.g. flow-based instead of packet-based operation.

An extension of IP-TNE [10] allows mixing of packet and fluid flows. The integration of fluid flows provides increased scalability. However, fluid flows can only generate synthetic background traffic. Since the flows model the transport layer themselves, they cannot be used to evaluate transport protocols. Hence, the approach is not fully transparent for layers below the application layer.

2.2 Node Virtualization

Resource partitioning for creating virtual nodes can be done on different layers. The spectrum ranges from emulating the entire hardware including processor architecture to process memory separation where each process runs in exclusive virtual memory as is provided by common operating systems. Here, we focus on concepts that are typically used for network emulation and support partitioning of our required resources: virtual machines and virtual routing.

Virtual Machine

Emulation approaches based on virtual machines (VMs) execute each vnode inside a virtual machine [1]. Protocol stacks are located on top of virtual network devices, which are connected to other VMs by a software switch with an uplink to other computers. Due to virtualizing the hardware interface, such approaches are fully transparent for the test subjects. In each VM runs a guest operating system which causes memory overhead. The same buffer cache entries may exist in possibly each guest as they exist in the host operating system or virtual machine monitor. A minimum Linux instance needs several megabytes of memory when executed in a VM based on the Xen [3] hypervisor. The same program libraries used in different VMs also need additional memory. While the memory overhead can be mitigated by content based page sharing, this in turn implies some computation overhead to calculate and compare page content. Network communication between vnodes based on VMs requires expensive context switches involving the hypervisor. Such overhead reduces the possible scenario sizes significantly. The authors in [12] report the faithful execution of 6 VMs on a pnode, whereas virtual routing is able to handle scenarios with up to 30 vnodes.

Virtual Routing

Virtual Routing (VR) [11] is based on a more lightweight virtualization than VMs. Only the protocol stack is virtualized and each process can be attached to a certain stack instance. In combination with approaches such as BSD jails [9], which partition the remaining required resources, virtual routing is transparent to the test subjects. The benefit of virtual protocol stacks is that only one operating system runs on each pnode, resulting in a minimal memory and computation overhead. Since all test application run under the same operating system, program libraries can be shared between vnodes. Communication between vnodes running on the same pnode even works without additional expensive context switches.

The comparison of VM based and VR based emulation approaches shows that VR implies less overhead and thus supports better scalability. The only remaining limiting factors for emulating large scale scenarios with hundreds of vnodes per pnode are the computation and network resources available in the hardware. Extending those limits is possible with time virtualization, which we discuss in the next section.

2.3 Time Virtualization

Several approaches increase scalability of network emulation by replacing real time with virtual time. The following sections cover different approaches for implementing and presenting virtual time.

Implementation Approaches

Two different ways of implementing time virtualization exist: virtual machines or modification of an existing operating system.

One of the topmost benefits of network emulation is the possibility of using real implementations and, therefore, virtual time must be transparent. Since the protocol stack also has to use virtual time, we cannot add the virtual time concept at the system call interface.

Gupta et al. [7] introduce the virtual time concept using virtual machines. By defining a time dilation factor (TDF), which is used to scale the time provided by the hypervisor to the virtual machine, it is possible to run a VM with an arbitrary virtual time. Since the test application runs inside the VM, it also runs with virtual time.

Another solution, used by Wang et al. [17], includes a real-time independent virtual time into the kernel. This virtual time is used by applications, the protocol stack, and their timers as time source. The main benefit of this approach is that no VM is required which results in less overhead. The missing VM abstraction causes an increased implementation overhead, because the virtual time concept has to be implemented throughout the entire kernel and not only at the small interface between hypervisor and the VM.

In comparison to our approach, both aforementioned approaches execute emulation scenarios on a single pnode and thus cannot leverage the resources of multiple distributed computers.

Approaches to Virtual Time Representation

Two extreme cases for the measurement and advancement of virtual time exist in the literature: slowing down real-time by a factor or discrete events.

The TDF (time dilation factor) introduced by Gupta et al. [7] scales real time by a constant factor. First, such an approach leads to the problem of selecting an adequate value for the entire experiment duration. Selecting too high a value wastes processing power and too low a value results in biased emulation results. Secondly, the load generated by the scenario varies over time. Hence, the TDF has to be selected for periods with maximum load and, therefore, hardware resource are not optimally utilized.

The authors of dONE (distributed open network emulator) [4] follow another approach. The idea is similar to discrete event simulation. Virtual time is dilated after processing events such as transmission of packets or after timers. Such an approach provides the emulator with unlimited processing power. dONE introduces a *time warp* operator to advance virtual time and skip periods where no events happen.

In case of a parallel discrete event approach, synchronization mechanisms are required to schedule events without causality errors. A causality error occurs when node 1 has a virtual time t_1 and node 2 has t_2 with $t_2 > t_1$ and node 1 sends a message to node 2 which has to arrive before t_2. Two basic approaches exist to handle this problem: conservative and optimistic [6]. The conservative approach prevents the existence of causality errors by executing events only if it can be guaranteed that no other node sends a message that has to be processed before. In case of the optimistic approach, events can be executed without limitations, but when a causality error is detected, the simulation state has to be rolled back.

When running real applications rollback-based approaches are very expensive, since each memory operation of the test subject needs to be logged and possibly recovered on a rollback. In the field of network emulation with event-based virtual time, only conservative approaches are feasible. Since the transmission times of frames are small, this results in a huge synchronization overhead. The authors of dONE [4] report to accurately emulate a TCP flow over a 100 Mbps link using FTP running at a third of real-time. Even running on slower hardware, our prototype accurately emulates a fully loaded 1 Gbps link using netperf while running at half of real-time, which is over a magnitude faster.

3 TVEE Approach

In the following, we describe our approach for improved scalable network emulation, called TVEE (Time Virtualized Emulation Environment), in three steps. First, we describe hybrid virtualization, which allows for creation of memory and network I/O efficient vnodes. Secondly, we introduce the concept of time epochs, which provide virtual time with low synchronization demand. Thirdly, we present mechanisms to emulate a network accurately in the presence of time virtualization.

3.1 Hybrid Virtualization

We now propose a hybrid virtualization approach combining virtual machines (VMs) and virtual routing (VR) to provide virtualization of time and nodes (Figure 1).

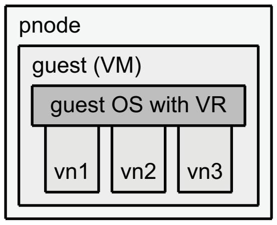

Figure 1. TVEE architecture: Multiple virtual routing instances inside one virtual machine

To limit the number of required code modifications for implementing virtual time, we make use of the virtual machine approach. A single VM per pnode provides virtual time. As shown in the evaluation, the memory overhead introduced by one VM is negligible and the CPU overhead can be handled by slowing down virtual time.

Since scalability is the topmost requirement, we use VR with namespace partitioning inside the single VM to virtualize nodes. As discussed during classification, the main benefits of VR are low per vnode memory overhead and efficient intra-pnode communication without additional context switches. Using the concept of namespace partitioning, each vnode has its own processes and filesystemes.

Hybrid virtualization allows the emulation of a large number of vnodes per pnode.

3.2 Time Epochs

During runtime of an experiment, the load on pnodes varies over time resulting in changing resource utilization. Minimizing the overall experiment runtime requires to maximize utilization of available hardware resources. Using virtual time based on discrete events maximizes resource utilization, however, at the same time introducing a high synchronization overhead. Alternatively, using a constant TDF requires no synchronization at experiment runtime but results in a low average resource utilization.

In order to maximize resource utilization without a high synchronization overhead, we introduce the concept of epochs. During an epoch, the TDF remains constant. All vnodes independently run with the same TDF. Assuming sufficiently accurate local clocks in the hosting pnodes, virtual time of vnodes remains synchronized without the need of any synchronization during an epoch. At an epoch transition, all VMs synchronously switch to a new TDF. Epoch based virtual time allows to minimize experiment runtime by solving the following optimization problem: selecting optimal TDF and epoch duration for a given load.

Solving this problem requires a definition of load. We are currently using the percentage of CPU time consumed by the virtual machine and host which is measured by the hypervisor. In contrast to this approach, which focuses on the CPU resource, an alternative approach may build on utilization of the physical network. Since overload of CPU or network biases emulation results, it stands to reason to use both load metrics. For the following description of our epoch based virtual time we assume the CPU based metric.

Overloaded pnodes may cause biased emulation results. Therefore, a mechanism to prevent overload is required. Whenever the load exceeds an overload warning threshhold t_o an upcoming overload is assumed and the TDF should be increased. Therefore, an overload report is sent to a central coordinator which instantly initiates an epoch transition. The coordinator ensures serialization of TDF changes in case of multiple overloaded pnodes causing multiple overload reports. However, optimal resource utilization also requires avoidance of underload. Therefore, a mechanism to detect underload and switch to a smaller TDF value is required. For this purpose, each pnode periodically sends load reports to the coordinator. If the load of all pnodes is below an underload threshhold t_u, the coordinator initiates a new epoch with a decreased TDF value.

There are various challenges associated with the algorithm outlined above. The first challenge is how to synchronously switch the TDF in a distributed emulation environment ensuring virtual time synchronization. Since pnodes are usually connected by a local area network, multicast can be used to inform pnodes to switch TDF at the same time. However, due to different loads on pnodes the time to process such a message in the protocol stack is not deterministic. To guarantee fast responsiveness, processing of TDF change requests should be performed in the hypervisor, which gets interrupted by a frame receive interrupt. Using this approach ensures epoch transitions with very low jitter.

A second problem we are currently investigating covers the question which TDF value to use. Dinda et al. [5] report epochal behavior of load, which encourages epoch based virtual time and also potentially allows load forecast of near future load [18]. Therefore, we investigate mechanisms to adapt the TDF value in future work. The key challenges are the avoidance of oscillations in case of dynamic load as well as fast responsiveness in case of sudden load peaks while at the same time minimizing the overall experiment runtime.

3.3 Network Emulation

After describing the virtualization of nodes and time, we now focus on the emulation of network properties in such a virtualized environment.

In each vnode, the protocol stack, consisting of network, transport, and application layer, is stacked on top of the virtual ethernet device. Connecting each virtual ethernet device with a bridge allows efficient communication between vnodes on same pnode. Joining the uplink port of the bridge with the pnode's ethernet device provides communication between vnodes running of different pnodes. Figure 2 shows the layers of our TVEE implementation.

In our implementation, we integrate the network emulation tool into the virtual network device driver. Placing the emulation tool inside the virtual routing instances enables back pressure on saturation of the emulated network connection as with real network devices.

The tool delays frames accordingly to the specified propagation delay and bandwidth limitation. It is also able to drop frames with a loss ratio, which can be configured for each sender receiver pair individually. This can be used to emulate wireless connections between nodes where the loss ratio depends on the channel quality between sender and

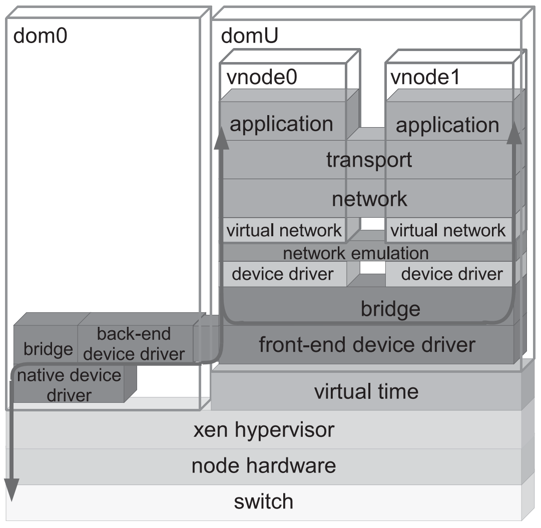

Figure 2. The layers of our TVEE implementation

receiver.

For each frame to process, the tool calculates the frame's transmission time, starts a timer to go off at this time, and places the frame in a delay queue. The frame is removed from the queue and transmitted when the timer expires. Since our emulation tool runs inside the virtual machine, the only available timers are based on the timer interrupt with a typical granularity of 1 ms. Generally it is possible to extend the VM to support hardware timers with a higher resolution such as APIC timers. However, this would require expensive context switches involving the hypervisor which we would like to avoid.

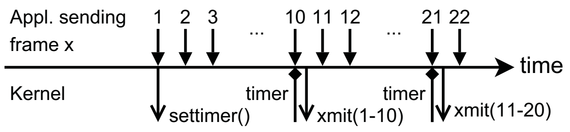

Figure 3. Emulation with interrupt triggered timers

For emulating high speed links (>Gbps) the granularity of interrupt triggered timers is too coarse. These links send more than a hundred frames per millisecond which results in a bursty transmission behavior. In Figure 3 the bursty behavior is visualized by example of an application sending a frame every 0.1 ms at a configured propagation delay of 1 ms. Each timer activates the transmission of ten frames. Such a behavior is not only unrealistic but it also leads to load peaks which prohibits load-aware TDF adaptation. To avoid these effects, we use a timer triggered by events [2] instead of interrupts.

The basic idea behind event triggered timers is to reuse existing system events to trigger the timer. TVEE uses frame-level sending and receiving events to trigger delayed frame transmission. Whenever a frame enters the emulation tool the delay queues are checked for frames to be transmitted.

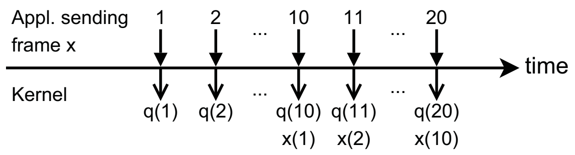

Figure 4. Emulation with event triggered timers (q = enqueue frame; x = transmit frame)

Figure 4 again shows an application sending a frame every 0.1 ms. However, here an event triggered timer is used for frame delays. After the first ten frames are enqueued, send requests of following frames trigger the transmission of previously delayed frames. Using this technique one frame is transmitted every 0.1 ms in this example.

The benefit of this approach is that increased link speed increases timer granularity, too. The granularity of the timers can be further improved by sharing the events of all emulation devices on a pnode. In case of periods where the duration between two frame-level events is higher than the configured delay, interrupt triggered timer are used as fallback solution. Since there are only a few frames to be delayed then, this does not result in a bursty transmission.

4 Evaluation

We evaluate our implementation in two steps. First, we show the accuracy of network emulation in our hybrid virtualization environment. Due to space restrictions we focus on the emulation of bandwidth and delay. Secondly, we show the scalability of TVEE as the paramount design criterion.

Xen [3] in version 3.1.0 using Linux (2.6.18) acts as foundation of our prototype implementation. Including OpenVZ [14] into Xen provides virtual routing with namespace partitioning. We extended Xen to provide guests running with virtual time as well as the virtual ethernet device driver of OpenVZ to provide configurable network properties.

All evaluation benchmarks are performed on Pentium 4 2.4 GHz PCs. Each PC is equipped with a Gigabit Ethernet adaptor and 512 MB of RAM. We divide the memory into 64 MB for dom0 and 429 MB for domU. The memory assigned to dom0 could be further decreased by running a minimal linux system. However, this constant amount does not affect the evaluation results.

4.1 Bandwidth Emulation

First, we measure if the emulation layer is able to enforce a configured bandwidth faithfully. Therefore, we set up a scenario with 2 connected vnodes in two variations. One variant uses a single pnode hosting both vnodes and the other uses two pnodes with one vnode each. We configure the link bandwidth with different values ranging from 64 kbps to 100 Gbps and no additional delay. To measure the maximum throughput of the link, we use the netperf tool [13] in UDP mode. It generates load according to configured send and receive buffers of 64 kB and an Ethernet MTU of 1500 Bytes.

As shown in Figure 5, the measured throughput corresponds to the configured bandwidth. Note that, due to the used hardware, for high speed links an emulation running at real-time is not possible. Therefore, we increased the TDF to avoid overloading of emulation nodes.

4.2 Delay Emulation

Next, we examine if the emulation tool faithfully reproduces configured delays. Again, the scenario consists of 2 connected vnodes. The link has a bandwidth of 100 Mbps and a variable delay between 1 ms and 100 ms. We use the ping tool to measure the round trip time between the vnodes. Variations of this scenario use one or two pnodes to host vnodes on same or different pnodes respectively. We also vary the TDF. As shown in Figure 6, the delays are emulated accurately.

4.3 Memory Consumption

As outlined during the discussion of the TVEE architecture, the scalability of an emulation solution heavily depends on the memory overhead. To evaluate the memory overhead of our approach we create a scenario with an increasing number of vnodes attached to the same network. Figure 7 shows the required memory usage.

The memory usage consists of two components. One constant amount for the base system including Linux kernel requiring about 27 MB and a constant per vnode memory usage of about 300 kB. In comparison, a vnode based on Xen requires at minimum 6 MB of memory [3]. As shown in Figure 7, the memory footprint increases linearly with the number of vnodes. This allows us to run over a thousand vnodes on a single pnode which is equipped with half a gigabyte of main memory. Please note that the main memory is shared by hypervisor, dom0 (64 MB), and virtual machine.

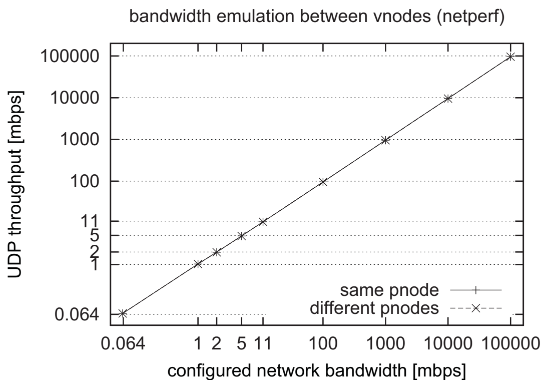

Figure 5. UDP throughput between two vnodes with different bandwidth

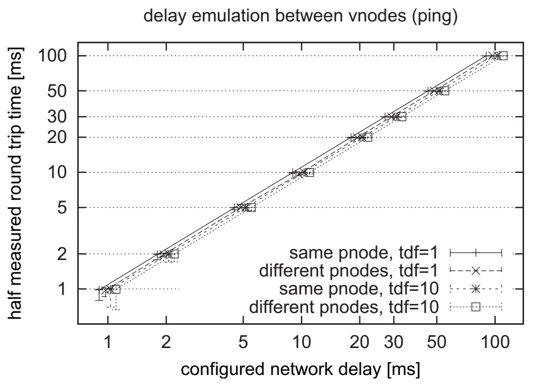

Figure 6. RTT between two vnodes with different delays

4.4 VNode/PNode Ratio

In the following, we use TVEE to measure a TCP flow through a chain of routers in an emulated network. The scenario consists of two pnodes, one hosting a router chain and the other a TCP sender and receiver. All links in the scenario are configured with a bandwidth of 100 Mbps and no additional delay. During the experiment different chain lengths are configured ranging from 4 to 253 routers. The maximum time-to-live of IP packets prohibits larger chains. We measure maximum TCP throughput while varying the TDF from -8 (running 8 times faster than real-time) to 64 (slowdown of factor 64). We use netperf with configured

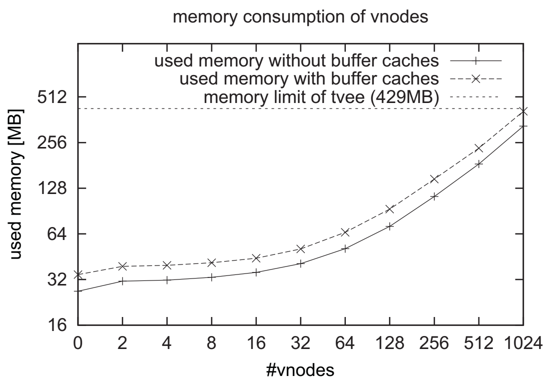

Figure 7. Required memory for increasing number of vnodes.

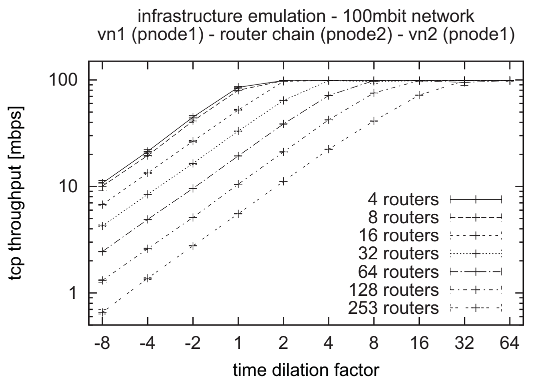

Figure 8. TCP flow emulation through a chain of routers

send and receive buffers of 64 kB to transmit data using sendfile for 10 s.

The results of this experiment visualized in Figure 8 show that TCP throughput of the system scales linearly with increasing TDF.

5 Summary

We introduced an approach to measure the performance of unmodified real distributed applications or protocols in a large controlled environment by combining hybrid node virtualization with epoch based virtual time. TVEE uses hybrid virtualization technique to efficiently map multiple virtual nodes (vnodes) to an emulation computer (pnode) with minimum overhead for network I/O and memory footprint per vnode. Time virtualization virtually increases pnode resources. Dynamically changing the time dilation factor

(TDF) minimizes experiment runtime and maximizes node utilization. In comparison to parallel discrete event simulation, our epoch based virtual time approach enables dynamic TDF with very low synchronization overhead. Using the concept of trigger based timers, TVEE is able to emulate high speed networks while reducing bursty frame delay.

Our evaluations showed that virtual nodes have a very low memory overhead of only about 300 kB. We showed that, using time virtualization, the performance of a pnode linearly scales with the TDF. This allows the emulation of scenarios with hundreds of vnodes per pnode. Additionally, links between vnodes can have bandwidths, which are magnitudes larger than the pnode's physical network bandwidth.

Currently, we are investigating mechanisms for monitoring the load of pnodes to provide the foundation of automatic load-dependent TDF selection. Based on resource monitoring, we will extend TVEE with mechanisms to automatically adapt the TDF to the current system load as well as to simultaneously switch epochs in order to optimize utilization of a distributed TVEE for long lasting experiments. Finally, we want to evaluate the advantage of event triggered timers.

References

[1] G. Apostolopoulos and C. Hasapis. V-eM: A Cluster of Virtual Machines for Robust, Detailed, and High-Performance Network Emulation. In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation (MASCOTS'06)*, pages 117–126, Monterey, CA, USA, Sept. 11–14 2006.

[2] M. Aron and P. Druschel. Soft timers: efficient microsecond software timer support for network processing. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, pages 232–246, Charleston, SC, USA, Dec. 12–15 1999.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 164–177, Bolton Landing, NY, USA, Oct. 19–22 2003.

[4] C. Bergstrom, S. Varadarajan, and G. Back. The Distributed Open Network Emulator: Using Relativistic Time for Distributed Scalable Simulation. In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06)*, pages 19–28, Singapore, May 24–26 2006.

[5] P. A. Dinda. The statistical properties of host load. *Scientific Programming*, 7(3-4):211–229, 1999.

[6] R. M. Fujimoto. Parallel discrete event simulation. In *Proceedings of the 21st Conference on Winter Simulation (WSC '89)*, pages 19–28, Washington, D.C., USA, 1989.

[7] D. Gupta, K. Yocum, M. McNett, A. C. Snoeren, A. Vahdat, and G. M. Voelker. To Infinity and Beyond: Time-Warped Network Emulation. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI'06)*, pages 87–100, San Jose, CA, USA, May 8–10 2006.

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

[9] P. H. Kamp and R. N. M. Watson. Jails: Confining the omnipotent root. In *Proceedings of the 2nd International SANE Conference*, 2000.

[10] C. Kiddle, R. Simmonds, and B. Unger. Improving Scalability of Network Emulation through Parallelism and Abstraction. In *Proceedings of the 38th Annual Simulation Symposium (ANSS'05)*, pages 119–129, San Diego, CA, USA, Apr. 4–6 2005.

[11] K. Kourai, T. Hirotsu, K. Sato, O. Akashi, K. Fukuda, T. Sugawara, and 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)*, pages 385–394, Bonn/Königswinter, Germany, Oct. 2003.

[12] S. Maier, A. Grau, H. Weinschrott, and K. Rothermel. Scalable Network Emulation: A Comparison of Virtual Routing and Virtual Machines. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC'07)*, pages 395–402, Aveiro, Portugal, July 1–4 2007.

[13] netperf. http://www.netperf.org, 2007.

[14] OpenVZ. http://openvz.org, 2007.

[15] G. F. Riley, R. M. Fujimoto, and M. H. Ammar. A Generic Framework for Parallelization of Network Simulations. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99)*, pages 128–135, College Park, Maryland, Mar. 24–28 1999.

[16] R. Simmonds and B. W. Unger. Towards scalable network emulation. *Computer Communications*, 26(3):264–277, 2003.

[17] S. Y. Wang and H. T. Kung. A New Methodology for Easily Constructing Extensible and High-Fidelity TCP/IP Network Simulators. *Computer Networks*, 40(2):205–315, 2002.

[18] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: the Network Weather Service. In *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing*, pages 1–19, San Jose, CA, USA, 1997.