Efficient and Scalable Network Emulation using Adaptive Virtual Time

Andreas Grau*†, Klaus Herrmann, and Kurt Rothermel
Universität Stuttgart, Institute of Parallel and Distributed Systems (IPVS)
Universitätsstr. 38, D–70569 Stuttgart, Germany
Email: {grau,herrmann,rothermel}@ipvs.uni-stuttgart.de
Telephone: +49 (711) 7816–236, Fax: –424

***Abstract*—Performance analysis and functionality testing are major parts of developing distributed software systems. Since the number of communicating software instances heavily influences the behavior of distributed applications and communication protocols, evaluation scenarios have to consider a large number of nodes. Network emulation provides an infrastructure for running these experiments using real prototype implementations in a controllable and realistic environment. Large-scale experiments, however, have a high resource consumption which often exceeds available physical testbed resources. Time dilation allows for reducing the resource demands of a scenario at the expense of the experiment's runtime. However, current approaches only consider a constant time dilation factor, which wastes a lot of resources in case of scenarios with varying load.**

We propose a framework for adaptive time virtualization that significantly reduces the runtime of experiments by improving resource utilization in network emulation testbeds. In this framework, resource demands are monitored and the time dilation factor is dynamically adapted to the required level. Our evaluation shows that adaptive virtual time in combination with our lightweight node virtualization architecture allows us to increase the possible scenario sizes by more than an order of magnitude and, at the same time, ensure unbiased emulation results. This represents an important contribution to making network emulation systems highly scalable.

***Index Terms*—network emulation, performance evaluation, virtualization, time dilation, adaptive virtual time**

I. Introduction

Two principal components of the software development process are functionality testing and performance evaluation. The behavior of distributed applications and communication protocols depends heavily on the number of communicating instances. Due to the complexity, an analytical evaluation of such systems is often infeasible. A commonly used approach is to evaluate the *software under test* (SUT) by means of a network emulation testbed. Using distributed emulation tools [21], [12] and a configurable network infrastructure, arbitrary topologies with configurable network properties can be created. The testbed consists of an arbitrary number of physical nodes connected by a high-bandwidth network. The big advantage of network emulation as opposed to the evaluation in a real network [6] and the usage of simulation tools [9], [20] is that the SUT does not have to be modified and can be evaluated in a repeatable fashion and in a controlled environment.

In order to support large scenarios in a network emulation, multiple virtual nodes can be executed on each physical node using *node virtualization* [1], [13]. However, the number of virtual nodes per physical node is limited, as mapping too many virtual nodes to a physical node overloads that node. Such an overload may bias evaluation results since the SUT experiences resource shortages that do not exist in a real execution environment. Using a virtual time [5] that is running slower than real time allows for reducing the system load and, therefore, allows to increase the number of virtual nodes per physical node. For example, a real 10Mbit/s network can be used to transport 100Mbit/s in the emulation if the *time dilation factor* is set to 10 (virtual time runs 10 times slower than real time). With this time dilation factor, a physical node using this real network can run 10 software instances in 10 virtual nodes where each consumes 10 MBit/s. Existing systems with a constant virtual clock rate choose a very conservative rate to prevent overload even when load peaks occur. Therefore, in scenarios with changing resource requirements, the system may experience considerable underload most of the time. The consequence is that the experiment's runtime is suboptimal.

In order to prevent overload situations and, at the same time, optimize the experiment runtime, we extend our *Time Virtualized Emulation Environment* (TVEE) [10], a network emulator based on node and time virtualization, by a mechanism to dynamically adjust the clock rate to the current load situation. Our evaluation results show that the adaptive time virtualization in combination with our lightweight node virtualization architecture, allows several hundred virtual nodes running on the same physical node without influencing the quality of the emulation results. Using the same hardware, we have shown in previous work [17] that other emulations approaches are only able to run in the order of 10 virtual nodes before results get biased. Therefore, our work represents a big step forward in terms of scalable network emulation. The resulting emulation framework can be used by the networking community in order to evaluate much larger networks and, thus, achieve more realistic results in many problem domains.

The remainder of the paper is structured as follows. In the following section, we first present the related work in the field of network emulation. In Section III, we discuss the

*Corresponding author.

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

Published in Proceedings of 18th Internatonal Conference on Computer Communications and Networks (ICCCN'09), pages 1 - 6, San Francisco, CA, Aug 2009.

http://dx.doi.org/10.1109/ICCCN.2009.5235306

architecture of our system with its main building blocks, node and time virtualization. In Section IV, we introduce the main contribution of this paper: the adaptive load-controlled virtual time approach. A detailed evaluation is given in Section V, and Section VI presents a summary, our conclusions, and an outlook on future work.

II. Related Work

Canon et. al. [5] where the first to introduce virtual time in emulations. However, no adaptive clock rate is used in their approach. Therefore, the clock rate has to be pre-configured to a value that ensures that no overload situation occurs throughout the experiment run. This leads to very conservative clock rates such that outside periods with load peaks, the system does not utilize the existing resources efficiently and, thus, the experiment runs much longer than necessary.

Hybrid systems combine the benefits of network simulation and network emulation, by connecting physical nodes [16] running real implementations to a simulated network based on parallel discrete event simulation (PDES) [9], [20]. In addition, these approaches use a combination of node and time virtualization [8]. However, a constant clock rate is used here too. The necessary synchronization in the simulation produces additional overhead.

Weingärtner et. al. [23] have proposed an approach to conservatively synchronize multiple virtual machines to a simulation framework. Here, an experiment is evenly divided into time-slices. The end of each slice constitutes a barrier to synchronize the VMs with the simulation framework. However, the VM-based node virtualization increases the overhead introduced by the synchronization schema.

Emulation of arbitrarily powerful virtual resources can be achieved by adapting the Linux protocol stack to use virtual time instead of real time. While Wang et. al. [22] use only a simulation framework running on a single physical node, dONE [4] uses a distributed simulation environment. In contrast to our system, dONE only supports testing of application layer implementations using the BSD socket interface.

All existing approaches either have additional synchronization overhead or only support a constant clock rate which both results in a suboptimal experiment runtime. TVEE solves these problems, by dynamically adjusting the clock rate to the current load of the system.

III. System Architecture

The foundation of our network emulation approach is the Network Emulation Testbed (NET) at the University of Stuttgart consisting of 64 nodes. The nodes are connected by an emulation network using a central, configurable gigabit switch and an additional control network. The emulation network can be partitioned using IEEE 802.1Q VLAN (virtual LAN) to form arbitrary virtual network topologies between the nodes. We have developed a tool called NETshaper [10], [12], to emulate network properties like bandwidth, delay and packet loss. Based on these components, we are able to run the SUT in a controlled environment. Since our network emulation approach provides a virtual network interface, we can evaluate distributed applications as well as communication protocols above data link layer.

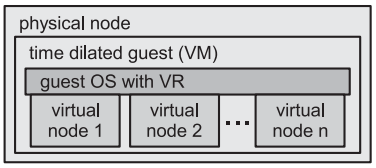

Fig. 1. TVEE architecture: Virtual Routing inside a virtual machine

To increase the number of virtual nodes (hosts, routers, switches) in the emulated network we use a node virtualization system that is based on *Virtual Routing* (VR) [15]. As opposed to *Virtual Machines* (VM) [2], [3], [7], Virtual Routing provides more lightweight virtualization. The processes that make up a virtual node all share the same operating system, but each of these virtual nodes has its own protocol stack, including sockets and routing tables. By virtualizing additional operating system components [14] like process name spaces and file systems, processes of different virtual nodes are clearly separated from each other. As shown in previous work [17], the VR-based approach has an order of magnitude better performance than the VM-based approach with respect to memory consumption and communication overhead. Thus, it is very lightweight.

Time virtualization is achieved by using virtual machines. By changing the interface between the virtual machine and the hosting operating system or virtual machine monitor, virtual time is transparently provided to every software running inside the VM [11].

The TVEE prototype architecture combines Virtual Routing for node virtualization and virtual machines for time virtualization as depicted in Figure 1. Multiple virtual nodes that are based on virtual routing run inside one VM that provides the virtual time. In turn, One of these virtual machines, in turn, run on each physical node in the emulation testbed.

IV. Epoch-based Virtual Time

To adapt the virtual clock rate to the resource demand of the experiment we introduce the *Time Dilation Factor* (TDF). Equation 1 shows how the virtual time (R_v) and the real time (R_r) are related by means of the TDF.

$$R_v = 2^{-\frac{TDF}{10}} \cdot R_r \tag{1}$$

In order to allow an efficient implementation using integer arithmetic, we use (in contrast to Gupta et. al. [11]) a logarithmic relation between the rate of virtual time and the real time. Using only integer arithmetics, we can adjust the rate of the virtual clock with a step width of about 7%. This granularity is sufficient for the adaptation algorithm that we will introduce in Section IV-B. In addition, without a logarithmic relation, this granularity depends on the virtual clock rate. For fast rates the granularity is coarse and it increases with slower rates.

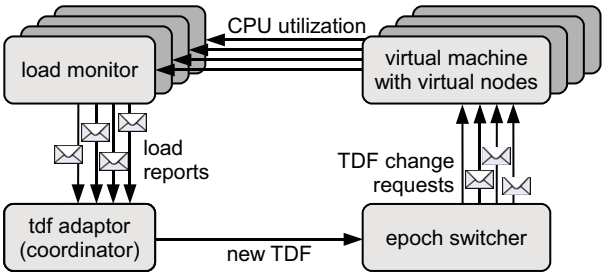

Fig. 2. TDF adaptation schema

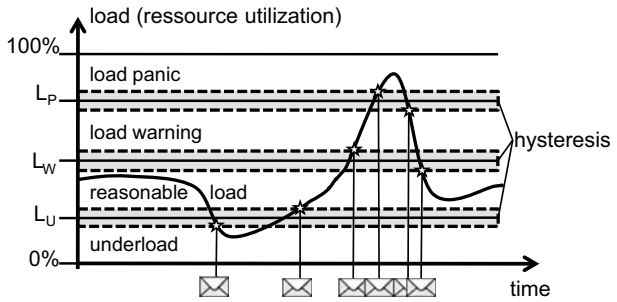

Fig. 3. Load Monitoring Thresholds

In order to perform the adaption of the TDF, we propose the concept of epoch-based virtual time [10]. The experiment is divided in epochs of different length where the TDF is constant within each epoch. Whenever the resource demand changes, an epoch switch is triggered to adapt the TDF.

Figure 2 shows the TDF adaptation schema. Each physical node of the emulation system runs a *load monitor* that monitors the node's load and reports it to a *central coordinator* who calculates the overall system load. The overall load is defined as the maximum over all individual node load values. This definition is chosen to ensure that no physical node is overloaded at any time.

Using the overall load, the coordinator determines a new TDF and initiates an epoch switch. The *epoch switcher* is used to distribute the new TDF to the physical nodes and to perform an epoch switch. In the following, each component (*load monitor*, *TDF adaptor* and *epoch switcher*) is discussed in detail.

A. Distributed Load Monitoring

The *load monitor* is used to measure the load of a physical node and report the load to the *TDF adaptor*. The virtual machine monitor (VMM) provides a per virtual machine statistic, which counts the number of used CPU cycles $c(t)$. Requesting this value at 2 points in time ($c(t_1)$ and $c(t_2)$) allows for calculation of the load $l = \frac{c(t_2)-c(t_1)}{t_2-t_1}$. To meter the time between the measurements with a sub-microsecond granularity, we use the time stamp counter register of processor (TSC), which is increased on every CPU clock cycle.

The length of the sampling interval has a large effect on the performance of load monitoring. Short intervals are required for a fast reaction to load changes, but also result in a large number of load reports. Transmission and processing of large amounts of load reports would overload the coordinator and, therefore, limit scalability. To limit the amount of load reports, we use 3 mechanisms: *adaptive sampling*, *threshold-based discretization* and *hysteresis-based state changes*. These mechanisms effectively reduce communication overhead for reporting substantially.

Adaptive sampling adjusts the length of the sampling interval (time period between two consecutive load reports) to the currently used TDF. For a higher TDF (slower virtual time) a longer sampling interval is chosen. The ratio behind this is that overload situations develop proportionally slower when the virtual time runs slower. Therefore, the sampling interval may be increased without taking the risk of missing any relevant change. The effect is that as a system increases in size (number of virtual nodes) and the load increases as a result, the message overhead resulting from load reports decreases. Therefore, we increase the sampling interval linearly with the TDF.

Threshold-based discretization maps the possible load values of a physical node to the 4 states *load panic*, *load warning*, *reasonable load*, and *underload* (see Figure 3) using 3 thresholds (L_P, L_W, and L_U). The load monitor determines the state locally and only in case of a state change, a load report is sent to the TDF adaptor. *Underload* indicates that there are unused resources and, therefore, virtual time could be accelerated. Analogous, the two states *load panic* and *load warning* signal that resource consumption is becoming too high. When the system is in one of these states, virtual time has to be slowed down. The two thresholds L_P and L_W are used to differentiate between slight and heavy *load*. The different reaction on these states is described in the next section.

Hysteresis-based state changes are used to avoid oscillation between two states which causes a high number of load reports. A state change is only triggered if load exceeds the threshold and its surrounding hysteresis range (see Figure 3).

B. TDF Adaptation

Based on the load reports, the TDF may need to be adjusted. The TDF adaptor achieves this adjustment by means of a very simple proportional feedback control mechanism that is shown in Algorithm 1. Whenever the system load is outside the *reasonable* range, the algorithm adapts the TDF to reach the *reasonable load* state. As long as the system load is in state *load warning* or *underload*, a small adjustment S_s is applied (added or subtracted) to avoid overshooting the *reasonable load* state. If there is a fast increase in load, this adjustment will not suffice and the system will eventually reach the *load panic* state. In this situation, a larger step size S_l is used for the adjustment in order to decrease the load quickly and avoid overload. If this results in an *underload* situation, the algorithm will gradually decrease the TDF again to speed up virtual time.

After each adjustment, the algorithm needs to wait for feedback from the load monitor to see whether the load is back in the state *reasonable load*. Due to the adaptive sampling of the load monitor, the time until the feedback arrives depends

input: $state$, TDF_{prev}

```
while true do
    if state != reasonable_load then
        if state = load_panic then
            setTDF(TDF_prev + S_l)
        else if state = load_warning then
            setTDF(TDF_prev + S_s)
        else if state = underload then
            setTDF(TDF_prev - S_s)
        end
        sleep T_s
    else if state = reasonable_load then
        setTDF(TDF_prev + S_s)
        sleep T_l
    end
end
```

Algo. 1. TDF Adaption Process

on the current TDF. Therefore, we dynamically adjust the waiting time (T_s) to the half of the used sampling interval.

In case of temporarily constant resource demands, the utilization can keep steady at any level between the L_U and L_W thresholds in the state *reasonable load*. For good resource usage, however, the system utilization should be near the L_W threshold. Therefore, we decrease the TDF in the *reasonable load* state, too. However, the speed of this adjustment is very low, through a waiting time T_l of an order of magnitude larger than the waiting time T_s. In combination with the hysteresis around the thresholds the oscillation around L_W, these adjustments introduce an insignificant overhead.

Evaluation shows that the introduced algorithm has a good reaction to changes of resource requirements despite the fact that it is rather simple.

C. Epoch Switching

After determining the TDF for the next epoch, a mechanism is required for propagating the new value to the physical nodes. To ensure a fast reaction on upcoming overload, the time between detecting the resource demand and the actual TDF change must be as small as possible. Since the time to compute the new TDF is negligibly small, we need to minimize the time for transmitting load reports and TDF change requests. In addition, realistic emulation requires all virtual clocks to run at the same rate at any time. Therefore, we need mechanisms to minimize the difference in propagation times of TDF change requests. A third problem related to epoch switching is the occurrence of message loss which cannot be detected in time.

We have developed a protocol for minimizing the propagation time of TDF change requests and load reports. The basic assumption behind this protocol is that all nodes are connected to a LAN. We are using the previously mentioned control network of the cluster. The delay of TDF change requests and load reports using this network consists of several components: network transmission delay, packet processing time in the protocol stack, and delay in queues. The time to transmit a frame in the network is insignificant because it is below 200μs and has a small variability. The processing time in the protocol stack is a magnitude below the transmission time and can be ignored as well. Most of the message delay is caused by waiting in egress and ingress queues of the physical nodes and the switch. In order to limit these delays, we are using priority queues based on *type of service* (TOS) of IP QoS and prioritize TDF change requests and load reports. A last source of delay are the hardware based FIFO queues inside the network interface cards (NICs). Since we cannot change theses queues, we are limiting the traffic on these interfaces to 95% of the link capacity to keep the queues empty. Using these mechanisms, the maximum packet transmission delay can be reduced below 2ms and message loss can be prevented with a very high probability.

V. Evaluation

In order evaluate the performance of the proposed system, we integrated the concepts for load monitoring, TDF adaptation and epoch switching into our prototype. The prototype, running on the NET Cluster equipped with 64 nodes (P4 2.4GHz, 512MB Ram, 1GBit NICs), is based on XEN [2] version 3.1.0 running Linux Kernel 2.6.18 inside a virtual machine (domU in XEN jargon) and inside the control domain (dom0). In addition, OpenVZ [19] is used to create virtual nodes inside the VM. The mechanisms for adaptive virtual time were implemented as Linux Kernel Modules (LKMs) running in dom0 to minimize latencies for epoch switching.

In previous work [10], we have shown that our emulation architecture is able to accurately emulate network properties like bandwidth and delay. By choosing an adequate TDF, we are able to emulate links between two virtual nodes with bandwidths between 64kbps and 100Gbps, as well as delays ranging from 1ms to 100ms. The virtual nodes in the following evaluation have an average memory overhead of 300KB.

The evaluation is structured as follows: First, we briefly discuss the chosen parameters for the load monitoring and the TDF adaptation. Then, we investigate the achieved resource utilization. Finally, we show how to evaluate the performance of a routing daemon in a large scenario using TVEE.

An extensive search of the parameter space using scenarios with different resource requirements has been performed to identify a configuration which generally minimizes experiment runtime and ensure unbiased results. The determined thresholds of the load monitor are: L_U=50, L_W=70, and L_P=90. The adaptive sampling interval ranges from 5ms for TDF=0 to 200ms for TDF=100. TDF adjustments with a step width S_s of 1 and S_l of 20 give best results for the TDF adaptation.

To quantify the achieved level of resource consumption, we are emulating a chain of routers routing 2 TCP flows. The test system consists of two physical nodes. On the first one, 2 virtual nodes are running the TCP sender and receiver of the first flow F_f. This flow is routed through the chain of routers with different lengths. The routers run on the second physical node. Additionally, one link of the router chain is used by a second flow F_b. The emulated network between the virtual

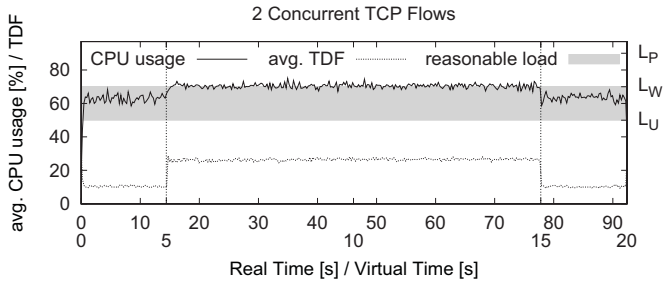

Fig. 4. Load-based TDF adaptation

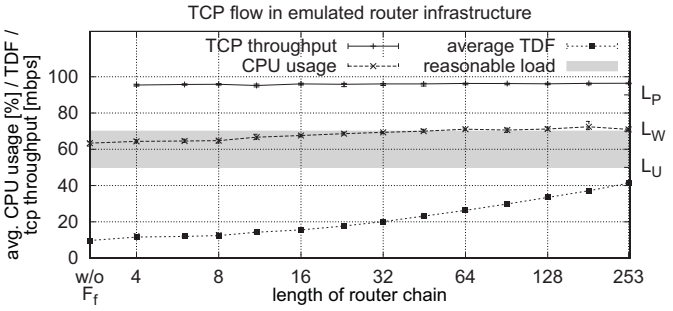

Fig. 5. Effectiveness of TDF adaptation

nodes has a bandwidth of 1Gbps except for the first and last link which have 100Mbit. During each experiment, we run the TCP flow F_b for 20s of virtual time. After 5s we run the flow F_f for 10s and measure the achieved throughput. In addition, the resource usage on both physical nodes is measured. For each router chain length, the experiment is repeated 50 times.

Figure 4 shows the CPU utilization of the physical node running a chain of 32 routers. The time axis has two sales: the upper scale is the real time and the lower scale the virtual time. Running only the flow F_b requires the system to run with a TDF of about 10 to keep the CPU utilization inside the *reasonable load* range. Running flow F_f between 5s and 15s of virtual time increases the resource requirements. In order to prevent overload, the system automatically adapts the TDF to a value of about 27. As flow F_f stops, the system adapts the TDF back to the original value.

Figure 5 shows the measured results for different numbers of routers, which are: the achieved TCP throughput, the load of the physical node running the router chain, and the average TDF. Although these measurements have different scales, we show them in a single graph to increase comparability. For comparison, we have also included the results for the experiment without flow F_f. The TCP throughput allows to rate the quality of the emulation by comparing the measurements with the TCP throughput in real environments. In the emulation as well as in measurements in real environments, TCP is able to achieve about 96Mbps throughput and, therefore, we can conclude that the emulation results are not biased.

As shown in the figure, for up to 8 routers the resource utilization mainly results from flow F_b. As the number of routers is increased, the resource requirements for flow F_f

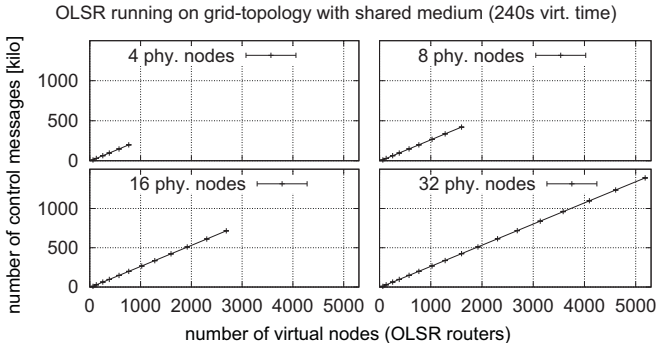

Fig. 6. Number of control messages sent by n virtual routers running OLSR

increase likewise. Since each router basically does the same, the load increases linearly with the number of routers. At a length of about 11 routers, the flow F_f consumes a significant amount of CPU and, therefore, the system needs to slow down the virtual time.

The gray area in Figure 5 marks the *reasonable load* range. For the experiment to exhibit minimal runtime, the resource utilization should be near the upper bound of the *reasonable load* range. As the Figure shows, the load of the physical node hosting the routers approaches this limit and stays below the threshold as desired. For shorter router chains, the low TDF results in a small sampling interval (see IV-A) which makes the system more sensitive to short load peaks. These load variations can cause false positives of *overload warning* messages and, finally, a temporary suboptimal TDF. However, the sensitivity is required to prevent overload situtations.

In the next evaluation experiment, we investigate scalability aspects with respect to the used number of physical nodes. In this experiment, we run an MANET (Mobile Ad Hoc Network) with an increasing number of virtual nodes. Each virtual node runs the OLSR protocol (Optimized Link State Routing) using the unmodified version of olsrd [18]. The nodes are arranged in a grid topology. Due to the configured transmission range, each virtual node can only directly communicate with its four neighbors. To verify that emulation results are not biased, we compare the number of control messages transmitted in an experiment. This number is depicted in Figure 6 for four different setups with 4, 8, 16, and 32 physical nodes and an increasing number of virtual nodes running uniformly distributed on them. As the number of virtual nodes is increases, the amount of control messages should increase linearly since each node emits such messages to its neighbors periodically. If results were biased by overload situations in the experiments, this would mean that the olrsd instances would not be able to emit the required control messages in time. Messages would be delayed or dropped. As a result, the linear increase would change and a kink would appear at the point where the number of virtual nodes gets too high. Figure 6 clearly shows no such sign of an overload situation throughout the entire range despite the increase in virtual as well as physical nodes.

Figure 7 shows the time required for running the OLSR scenario with different numbers of physical nodes. For low

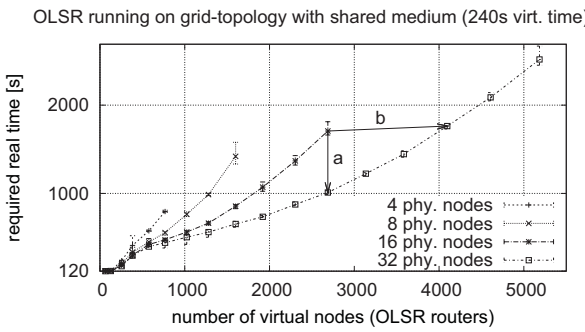

Fig. 7. Required real time to emulate scenario with n virtual routers running OLSR

numbers of virtual nodes (< 300), increasing the number of physical nodes does not produce a notable effect as these experiments can easily be run using 4 and 8 physical nodes. However, the figure shows that, for a larger number of virtual nodes, the resources of all physical nodes are efficiently exploited. For example, with 2700 virtual nodes, doubling the resources from 16 to 32 physical nodes can reduce the required experiment time by about 40% if the number of virtual nodes is kept constant (arrow a). Conversely, through doubling the resources, the number of virtual nodes (and, thus, the size of the scenario) can be increased by 50% without increasing experiment time (arrow b).

VI. Conclusion

The TVEE emulation system builds on a combination of node and time virtualization. This virtualization schema provides a highly scalable infrastructure for the evaluation of distributed applications and communication protocols.

The proposed node virtualization techniques allow running up to a few hundred virtual nodes on each physical node. Our time virtualization approach allows for a dynamic adaptation of the virtual time perceived inside the emulated software in order to react to the current amount of resource consumed by the SUT. As we have shown, by slowing down virtual time, overload situations can be avoided and, therefore, unbiased experiment results are ensured. Moreover, the resources of a testbed are utilized effectively such that experiment execution times are minimized.

Our evaluation shows that, with TVEE, the available CPU and network capacity no longer limits the possible scenario sizes. The scenario size is only limited by the available memory and the permissible execution time of an experiment. This is an important step forward in terms of emulation scalability. Therefore, TVEE represents a general tool allowing network researchers to explore large scenarios.

The mapping of virtual nodes to the physical nodes heavily influences the achieved resource utilization on the individual physical nodes. In our future work, we will investigate automatic placement schemas for virtual nodes in order to achieve load balancing. A second field of future research is the efficient usage of multi-core processor architectures for network emulation.

References

[1] G. Apostolopoulos and C. Chasapis. V-eM: A Cluster of Virtual Machines for Robust, Detailed, and High-Performance Network Emulation. Technical Report 371, ICS-FORTH, Greece, Jan. 2006.

[2] 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, 2003.

[3] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the USENIX 2005 Annual Technical Conference, FREENIX Track*, pages 41–46, Anaheim, CA, USA, Apr. 10–15 2005.

[4] C. Bergstrom, S. Varadarajan, and G. Back. The Distributed Open Network Emulator: Using Relativistic Time for Distributed Scalable Simulation. In *Proc. of the 20th Workshop on Principles of Advanced and Distributed Simulation*, pages 19–28, 2006.

[5] M. D. Canon, D. H. Fritz, J. H. Howard, T. D. Howell, M. F. Mitoma, and J. Rodriquez-Rosell. A Virtual Machine Emulator for Performance Evaluation. *Commun. ACM*, 23(2):71–80, 1980.

[6] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):00–00, July 2003.

[7] J. Dike. A user-mode port of the Linux kernel. In *Proc. of the 5th Annual Linux Showcase and Conference*, Oakland, California, Nov 2001.

[8] M. Erazo, Y. Li, and J. Liu. SVEET! A Scalable Virtualized Evaluation Environment for TCP. In *Proceedings of the 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom'09)*, 2009.

[9] R. M. Fujimoto. Parallel Discrete Event Simulation. In *Proceedings of the 21st conference on Winter simulation (WSC'89)*, pages 19–28, 1989.

[10] A. Grau, S. Maier, K. Herrmann, and K. Rothermel. Time Jails: A Hybrid Approach to Scalable Network Emulation. In *Proc. of the 22nd Workshop on Principles of Advanced and Distributed Simulation (PADS'08)*, 2008.

[11] 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 ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI 06)*, pages 87–100, 2006.

[12] D. Herrscher and K. Rothermel. A Dynamic Network Scenario Emulation Tool. In *Proc. of the 11th International Conference on Computer Communications and Networks (ICCCN 2002)*, pages 262–267, 2002.

[13] X. Jiang and D. Xu. vBET: a VM-Based Emulation Testbed. In *Proc. of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research (MoMeTools'03)*, pages 95–104, 2003.

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

[15] 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 *Proc. of the 28th Annual IEEE International Conference on Local Computer Networks (LCN'03)*, pages 385–394, 2003.

[16] J. Liu. Immersive Real-Time Large-Scale Network Simulation: A Research Summary. *In Proc. of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*, pages 1–5, April 2008.

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

[18] olsrd. http://www.olsr.org, 2009.

[19] OpenVZ. http://openvz.org, 2009.

[20] G. F. Riley, R. M. Fujimoto, and M. H. Ammar. A Generic Framework for Parallelization of Network Simulations. In *Proc. of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 128–135, 1999.

[21] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41, 1997.

[22] 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.

[23] E. Weingärtner, F. Schmidt, T. Heer, and K. Wehrle. Synchronized Network Emulation: Matching prototypes with complex simulations. In *Proceedings of the First Workshop on Hot Topics in Measurement & Modeling of Computer Systems (HotMetrics'08)*, Annapolis, MD, 2008.