2025 36th International Teletraffic Congress (ITC 36)

Continuous Integration for Networks Supporting Low-Latency Using Hybrid Network Emulation

Florian Wiedner, Dominik Kreutzer, Jonas Andre, Georg Carle
Department of Computer Engineering
Technical University of Munich
Garching by Munich, Germany
{florian.wiedner, dominik.kreutzer, jonas.andre, carle}@tum.de

***Abstract*—Enabling continuous integration (CI) cycles for network protocols and services poses a significant challenge due to the necessity of building complete and complex networks for testing and verification. This process demands robust simulation, emulation, or a variety of hardware resources.**

For non-latency, throughput-sensitive services that deal with best-effort traffic, tools like Mininet or ns3 can be utilized effectively. However, latency-sensitive applications require verification in circumstances that closely resemble real-world environments. Tools such as ns3 operate at an abstraction level that is too high to accurately reflect reality, while original Mininet, by relying on virtual Ethernet pairs, tends to lack realistic latency.

To address this challenge, we propose using Mininet as a standard and reproducible API, enhanced with Single-Root-IO-Virtualization (SR-IOV)-based connections when stability and lower latencies are paramount. This approach enables us to test and verify working configurations in a straightforward, non-hardware-supported environment, allowing the final stages of our CI process to progress towards more stable products without the necessity to adapt scripts or configurations due to reusing the same API for different underlying technologies. Our findings demonstrate that incorporating SR-IOV into network emulation can potentially double the usable bandwidth and significantly enhance both stability and latency.

***Index Terms*—low-latency, throughput, single-root-io-virtualization, emulation**

ITC Library

I. Introduction

Continuous Integration (CI) offers significant advantages towards developing, extending, and improving tools using testing and verification to identify problems early [1]. Adapting network protocols and services is even more challenging as failures can impact other devices and services in the network.

The complex nature of networks causes challenges when analyzing protocols and services in CIs as the inter-working of components such as routers and firewalls must be considered. This complexity results in a limited testing coverage performed in today's networks. Digital Twin Network (DTN) provides the opportunity to utilize a digital replicate of the actual network to perform tests and verification without disturbing the original network, similar to other research areas [2].

Several tools provide a DTN based on either mathematical models [3]–[6], simulations [7], [8], or emulations [9], [10]. DTNs based on mathematical models provide, similar to simulations, an abstraction of the network addressing specific aspects of the real network based on their use case without the ability to test or verify application code directly. However, network emulations such as Mininet using Linux on-board tools allow to rapidly build an entire network to test and verify services and protocols automatically without a physical network [9]. Especially towards latency, Wiedner et. al. [11] compared in a previous work Mininet to a virtual machine (VM)-based approach with links based on Single-Root-IO-Virtualization (SR-IOV), which extends the PCI specification enabling splitting one NIC into multiple lightweight PCI devices, showing that the results are neither realistic nor stable.

This paper analyze a hybrid network emulation tool based on Mininet with SR-IOV and VMs. We aim to perform a controlled comparison using an external measurement infrastructure to provide insights towards analyzing latency-stable and high throughput environments. We provide:

1) A extended Mininet framework with SR-IOV and VMs.
2) An extended performance analysis.
3) Performance enhancements for sensitive network tests.
4) Recommendations on using the different types in common scenarios such as CI or DTN.

The paper is structured as follows: Section II provides background information and the current state of the art, followed by challenges when using CIs for low-latency network services in Section III. We outline our methodology to extended the Mininet framework in Section IV. Followed by the measurement setup in Section V and the evaluation in Section VI. We finalize the paper with limitations, recommendations, reproducibility, and conclusion in Sections VII to X.

II. Background and Related Work

This section presents an analysis of relevant literature in the fields of network simulation and emulation, DTN, and CI.

A. Network Simulation and Emulation

One important aspect of network simulation and emulation is the opportunity to test and verify network behavior in a closed, controlled environment, allowing rapid testing at different abstraction levels. It gives the flexibility of rebuilding arbitrary network topologies, including complex network components and their inter-working [12]. Simulation uses mathematical models, whereas emulation uses abstraction tools to represent the physical network.

Several tools providing general-purpose network simulation and emulation facilities are available, such as, e.g., OMNet++, NS3, or Mininet [13]–[15] providing a rich range of features. Wiedner et al. [11] show that Mininet is not designed for realistic tail-latency and throughput. They propose using cabled SR-IOV interfaces to replace virtual links with highly optimized VMs as network nodes and show that they can achieve realistic low and stable latencies but did not integrate it in an automatable framework needed in CI. Furthermore, simulation tools such as OMNet++ [13] and NS3 [14] are based on abstract mathematical models of the real world to describe the network in discrete events.

Mininet [15] uses Linux on-board features and a Python API to built a network by isolating different nodes using Linux namespaces, and using virtual Ethernet pairs (veth) as links. This emulation architecture allows using Linux standard network applications in real-time. Furthermore, Yan and Jin [16] are overcoming the challenge of reduced bandwidth capabilities in Mininet when the experiment load is higher than the host's physical resources by adding a virtual time to the nodes improving the results' realism. However, it improves tail-latency far over reality as it does not consider the time-wise correlation of events on different nodes. Containernet, a Mininet fork, provides Docker containers as network nodes and features such as CPU or memory isolation using the Mininet API [17]. With Containernet 2.0 Peuster et. al. [18] added support for integrating VMs into Mininet, using veths as connections based on Containernets API. As VMs allow to use applications requiring kernel access, our focus in this paper lies on extending the approach of Containernet 2.0 by adding VMs and hardware-based connections to reuse the existing API for tail-latency analysis.

B. Single-Root Input/Output Virtualization

One extension to utilize the flexibility of providing arbitrary topologies without the requirement of cabling and supporting processing packets on real-hardware to improve performance is SR-IOV. SR-IOV extends the PCI specification, enabling the splitting of one physical function (PF) into multiple virtual functions (VF)—lightweight PCI devices. Each VF has a separate send and receive queue distinct from the PF queues to reduce overhead and involvement of the OS in packet processing [19]. Liu [20] and Dong et al. [19] have both demonstrated that SR-IOV is improving both bandwidth utilization and latency in comparison to drivers such as virtio to distribute the packets from one physical link into multiple virtual devices. Using SR-IOV, it is possible to achieve line rate in each node without the need to pass through one physical network interface card (NIC) per node. Similar to [11] are we utilizing SR-IOV in combination with VLAN-IDs to provide traffic of multiple link connections over one cable.

C. Digital Twin Network

Network emulation can be the basis for DTNs providing a virtual twin of the physical infrastructure to analyze and optimize operations and behaviors [2]. DTNs can be further built using graph-neural networks [5], simulation, or AI [4] Tang et. al. [21] outline in their research the need for delay-sensitive traffic within DTNs as several potentially analyzed service-level agreements require this. Our approach allows to use Mininet as latency-sensitive DTNs.

D. Continuous Integration

CI is the central concept to allow automatic testing and verification of systems before deployment coming from the Extreme Programming development process [1]. CI is an automated process executed on each commit in a software repository, providing a direct feedback loop to the developers. It improves code quality through testing and enforces code styling [22]. CI includes automatic build and test software before new features or bugfixes are deployed into production, reducing the human effort in testing the software and reduce the error rate of deployed software.

Vucnik et al. [23] describe the use of CI for wireless communication as challenging due to their complex nature. A unique system is needed integrating web technologies and OS virtualization in existing 5G testbeds to enable CI for wireless communication. DigSiNet [24] is an example for a container-based DTN used for CI in a testbed infrastructure to predict failures in network configurations. Our paper supports the idea of network CI with a framework for automated testing of low-latency and high throughput services.

E. Low-latency in Virtualized Systems

Virtualization enables resource sharing and flexible, on-demand provisioning of resources, especially in the context of DTN and CI this is a requirement due to their fast-changing and adapting nature. Two commonly used architectures for virtualization are hypervisor- and container-based. VMs are Hypervisor-based virtualizations isolating the complete OS, including the kernel. Whereas containers are called lightweight virtualizations using a shared kernel [25]. Several studies such as [20], [26]–[29] analyzed latency in both containers and VMs providing valuable insights into the differences between both techniques. Daichendt et. al. [30] analyze the influence of multiple containers on each other, showing that the influence of the nodes, especially towards latency, is significant. Wiedner et al. [11] showed that using several OS-level optimized VMs and SR-IOV improves the end-to-end latency and stability significantly [11]. We extend Mininet with technologies proposed by [11] to improve latency stability.

III. Continuous Integration and Low-latency of Network Services

CI provides a long list of advantages towards continuous feature development in software systems [1]. However, when analyzing networks and their services, and the communication structure, it is a challenge to integrate a CI in a near-to-reality environment [17]. Emulations allowing rapid prototyping in entire networks are needed to analyze network services and protocols in a DTN enabling the use of Linux networking software directly. This design improves the possibility of

ITC Library

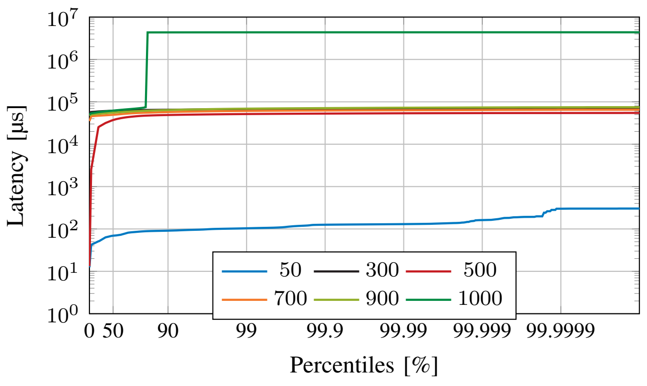


Figure 1: Latency HDR diagram across various rates (kpackets/s) send through original Mininet.

testing services that require different parts distributed within the network. To allow executing software requiring kernel access, Peuster et al. [18] extend Mininet with the opportunity to use VMs as Mininet nodes. Mininet's flexibility allows for the rapid building of general-purpose networks automatically.

To illustrate current latency and throughput issues, we perform measurements with constant-bitrate (CBR) traffic from an external load generator with a packet size of 84 B and a seven nodes line-topology with nodes directly connected to each other without switches in Mininet using a vanilla Debian Bookworm OS. The maximum stable packet rate in our setup on Mininet is 270 kpackets/s. With rates below 270 kpackets/s, we achieve a stable packet rate processed through Mininet. Figure 1 shows the latency results at different rates from 50 kpackets/s until 1000 kpackets/s in a high dynamic range (HDR) diagram with both axes logarithmic. The x-axis presents the percentiles of the latency shown on the y-axis. Similar to previous results such as [11] is the average and tail-latency reaching over 80 ms in comparison to experiments using real cabled connections. Additionally we observe 20 % packet loss with 500 kpackets/s in this scenario and as the packet loss increases, the worst-case latency stays mostly stable due to dropped packets reducing the load on the memory-based links. This already shows the importance of improving the potentials of Mininet.

This is especially problematic when analyzing latency-sensitive networking services and applications as the latency values are mainly caused by high load, not the analyzed application. When network services have strict requirements on the traffic such as low-tail-latencies or deterministic bandwidth, then virtualization is a challenge as Gallenmüller et al. [27] outline. They show that most latency outliers in virtualized systems are caused by interrupts due to the higher cost of interrupting virtualization. Typically, such systems require a strict latency until the 99.9999th percentile; rare outliers in higher percentiles are not considered [27]. Especially when using namespaces for isolation the influence of the core on which the process is running and the interrupt affinity is significant compared to bare-metal systems [30].

Stable latencies are essential to model the differences between emulated and real networks, analyze the latencies, and confirm that the processing delay in combination with the network traffic, for example, holds the requirements defined for the protocol or application. Based on the results shown in Figure 1, the paper aims to analyze the potentials of using a defined API such as Mininet and the potential to rapidly build networks and allow rapid prototyping and testing of network applications in combination with state-of-the-art optimizations to allow analyzing low-latency, deterministic network applications, services, and protocols. Our motivation example in Figure 1 shows that currently, the latency in Mininet is reaching much earlier than realistic network appliances an overload and therefore having higher packet loss and a high gap between lower and higher percentile latencies making it unrealistic to be used in CI automation for low-latency network services such as Time Sensitive Networking algorithms to be tested. Identifying if it is possible to achieve both flexibility and low latencies is the main target of this paper, concluding that finding a way to utilize a standard API to support easy integration of CI automation and optimizing towards deterministic and reliable latency is essential.

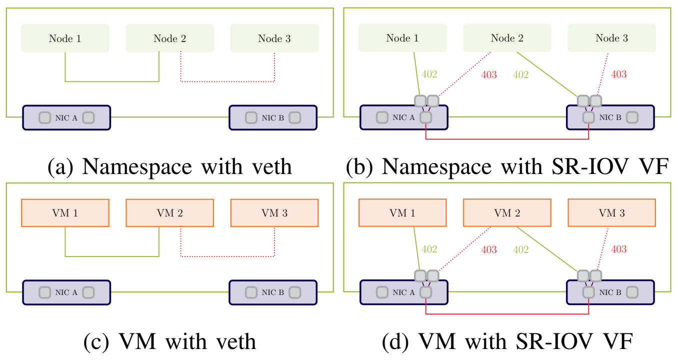
IV. Methodology

We present our methodology to extend Mininet towards more reliable but still rapid prototyping when low-latency or deterministic bandwidth are required. We extend Mininet with VMs similar as done previously in Containernet 2.0 [18]. Additionally, we utilize SR-IOV as hardware connections based on Wiedner et al. [11] to provide hybrid emulation of topologies. Furthermore, we use a small set of optimizations on the hypervisor as their usage depend on the control over the hypervisor and corresponding bootparameter are those not integrated into the extended Mininet version.

With extending Mininet, we aim to provide a realistic CI environment for low-latency or deterministic bandwidth network services and protocols. Additionally, we show that using our approach, we achieve a flexible and scalable solution to emulate complex topologies and provide performance benchmarks of network services as part of DTNs or CIs that can be extended towards a development or measurement platform.

Using VMs, the range of available network services and protocols is extended as kernel access is available as required in some applications. Carefully optimized VMs show similar performance towards bandwidth and tail-latency as Linux containers but require more overhead in resource usage as examined by Wiedner et al [26]. Figure 2 shows all four variants analyzed in this paper. The abstract architecture neglects the Mininet python API and the hypervisor. In Figure 2a, the traditional Mininet approach with namespaces and veth links is presented as baseline for our extended Mininet framework. Our methodology includes the following variations: combining VMs with veth based links or replacing the links with SR-IOV-based VFs using Virtual LAN (VLAN) IDs in both variants as shown in Figures 2b and 2d to distinguish the links on the

ITC Library



(a) Namespace with veth (b) Namespace with SR-IOV VF

(c) VM with veth (d) VM with SR-IOV VF

Figure 2: Node-link type combinations analyzed—link label representing VLAN IDs.

shared cable. Using this approach, we can use switches within Mininet, and the connections are working for both multicast and unicast connections as explained by Wiedner et al. [11].

To abstract the node type from the user in our approach, we added a new node type `VMNodeMixin` as basis extended towards switches or hosts. This class translates between QEMU, a generic engine virtualizing machines, and Mininet API. In our methodology, QEMU, combined with kernel virtual machines (KVM), utilized as hypervisor for VMs. This allow us to use the same Mininet scripts for VMs and namespace nodes as well as the flexible change between node types. To enable adding interfaces in Mininet, they are added after the node is booted, using a hot-pluggable PCI Express to PCI bridge. Interfaces are either added directly in passthrough mode or using MACVTAP devices as intermediate if direct passthroughs are not supported. VM-based nodes expose multiple configuration options to let the user configure the number of CPUs, memory, the path to the kernel, initrd and file-system images, and CPU pinning to allow optimizations [27]. The communication between Mininet and the VM node is solved using a Unix socket passed to the guest. A Mininet shell is spawned inside the VM connected over the virtual console to the Mininet process on the hypervisor. Adding a new node type makes the usage simple and extensible.

To improve the performance, we want to support hardware interfaces, in our case, SR-IOV VFs, by defining the interface on which VFs are available. The corresponding number of VFs per interface and pair-wise VLAN IDs must be configured before. This approach reduces the needed adoptions in Mininet when the hardware changes, as the configuration mask including spawning of VFs is vendor-specific. Hardware links are added as a new link type `HwPair` expecting a pair of already existing interfaces connected directly or, for example, using SR-IOV and VLAN-IDs for separation. To simplify this process, a global list can be provided to Mininet with pairs of interfaces each representing a possible direct connection. Access to the code is described in Section IX.

Our methodology can be used with most Mininet forks. We implemented our prototype using the original Mininet as the feature set is more condensed than forks such as

ITC Library

Table I: Latency optimized boot parameters for Host OS.

Parameter	Value	Description
irqaffinity	0	Interrupts on specific core
idle	poll	Poll mode when core idle
tsc	reliable	Rely on TSC without check
mce	ignore_ce	Ignore corrected errors
audit	0	Disable audit messages
nmi_watchdog	0	Disable NMI watchdog
skew_tick	1	No simultaneous ticks for locks
nosoftlookup		Disables logging of backtraces
nosmt		Disables hyperthreading

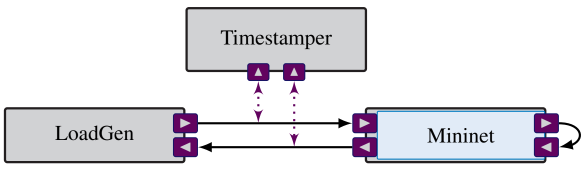


Figure 3: Measurement-setup derived from Wiedner et al. [26].

Containernet. Therefore, adding extensions such as SR-IOV and VMs is more straightforward. Moreover, mixing different types of nodes and links in one experiment is possible by using different node and link classes for the definition of the topology. The following section outlines how our measurement setup enables to precisely evaluate the performance of Mininet and our extensions and reduces the influence of the load generation and measurement onto the results.

V. Measurement Setup

As outlined in Section II, interrupts, among other features like the sleep state of the CPUs and memory copies, are the highest influence factor on forwarding latency within one machine. With building an entire network topology on one machine, the influence is even higher. In several studies such as [26], [27], [31] network latency optimizations are explained, and the lowest latencies are achieved when using a combination of OS-level optimizations, user-space networking, and direct access to the NIC. In our setup, we use a baseline of optimizations such as isolation techniques, disabling watchdogs, and a list of additional optimizations to reduce the impact of audition and sleep states of the CPU on our measurements [26]. The selected parameters are shown in Table I. These optimizations include disabling energy-saving mechanisms, moving the interrupt affinity to a specific core, and turning off audit messages. Interrupts affinity can be set to a specific core, and logging of backtraces can be reduced to improve latency. For the VM, we analyze two different modes, a non-optimized with only the baseline optimizations and an optimized version with pinned and isolated cores to reduce the influence of interrupts on the latency for reducing the overhead caused by using full virtualization.

To allow for precise measurements without influencing the Device-under-Test (DuT) running Mininet with the measurement process itself, we use a three hardware-machine-based setup as proposed by Wiedner et al. [26]. Figure 3 depicts the structure based on a load-generator (LoadGen), a timestamping machine (timestamper), and the DuT running the software to analyze. The LoadGen uses an Intel Xeon Silver 4116 CPU, 192 GB RAM, and a dual-port Intel 82599ES 10-Gigabit SFP+ NIC connected to the DuT using optical fibers. We use the second machine as timestamper connected to the fibers between DuT and LoadGen with passive optical terminal access points (TAPs) to ensure high precision measurements per packet at line rate. These TAPs add the same constant delay on both sides, which can be neglected as it is automatically removed in the latency calculation consisting of the time between the two measured timestamps. The timestamper is equipped with an AMD EPYC 7542 32-Core Processor, 128 Gbit of RAM and a dual-port Intel E810-XXV 25 Gbit/s NIC flashed to 10 Gbit/s. The hardware timestamping of the Intel E810 NIC offers a precision of 1.25 ns [32].

The DuT is equipped with an AMD EPYC 7551P 32-Core Processor, 128 GB RAM, and 2 $\times$ Intel X710 10GbE SFP+ NICs with one port of each NIC is linked to the LoadGen. The other two ports are linked together and used as a loop cable for the two setups with SR-IOV-based links as shown in Figures 2b and 2d.

The setup shown in Figure 3 allows a precise analysis of packet processing tail-latency to focus on comparing the different configurations with each other instead of the different measurement execution influences. We use MoonSniff [33] of MoonGen on the LoadGen and timestamper to transmit and record minimally sized packets with 84 B. The size of 84 B is the minimum packet size, allowing a large enough ID to transmit each packet as part of the payload. If not otherwise specified, we are using CBR traffic in all experiments.

We further use Debian Bookworm 11 (kernel 6.1) as both hypervisor and VM OS to utilize stable software library versions on a currently available OS version. Our Mininet version and QEMU are running inside the DuT, processing packets arriving from the LoadGen and sending them back to the LoadGen after traversing through the topology deployed with Mininet. The QEMU API and emulator is used for executing KVM VMs. For a deeper understanding of the scripts and code executed, we redirect readers towards Section IX.

ITC Library

VI. Evaluation

To assess the proposed extended Mininet methodology, we first present the scenario used for our evaluation, conduct baseline measurements, and analyze this scenario's potential usable bandwidth and latency. Following this, we compare an optimized version with an unoptimized version based on VM measurements. We conclude our evaluation by analyzing a scenario with burstiness in traffic and its influence on the variants. The results obtained are used to offer the reader tailored recommendations for various usage scenarios.

A. Scenario

To evaluate the Mininet topology, we select a configuration that is both straightforward and capable of stressing the system using CBR traffic. Additionally, the scenario is designed to illustrate the interactions among multiple nodes within the same system.

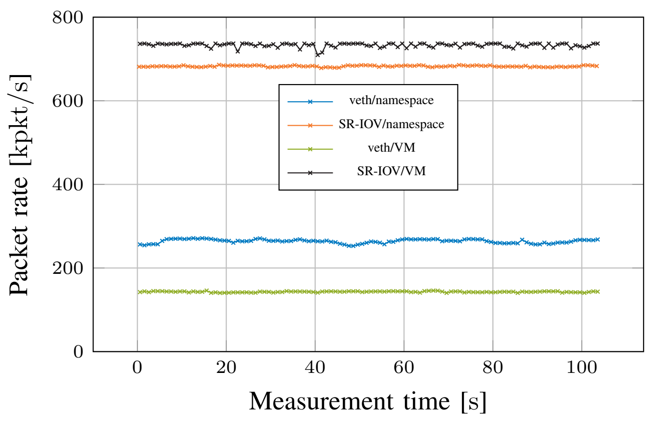


Figure 4: Packet rate over time after passing through Mininet—LoadGen transmit CBR traffic with 700 kpackets/s.

We utilize a line topology comprising seven nodes, labeled `h1` through `h7`, where `h1` is responsible for receiving traffic from the LoadGen, and `h7` transmits the traffic back to the LoadGen. No further nodes are involved, the nodes are directly connected to each other reducing the complexity for simpler understandability of the results.

To facilitate traffic forwarding among the different nodes, all hosts employ Linux routing and forwarding capabilities, routing traffic based on their respective routing tables to the next hop in line. This scenario effectively stresses the application with high packet rates and yields valuable insights.

We investigate a selected range of packet rates—50, 300, 500, 700, 900 and 1000 kpackets/s—across all four variants depicted in Figure 2. Our aim is to analyze how the different variations respond to varying packet rates in relation to latency, as well as to identify the maximum packet rate that each variation in the same scenario can effectively process.

B. Baseline measurement

To understand the changes identified in the evaluation of the newly added variants using SR-IOV and/or VMs in Mininet, we first evaluate a baseline with the original, not adapted Mininet. For identifying the bandwidth bottleneck, the packet rate is displayed in Figure 4 over the time that was received after transmission through Mininet with CBR traffic. We selected 700 kpackets/s as rate to present in the diagram over time after analyzing the data in Figure 4 as blue line. 700 kpackets/s is higher than Mininet can handle the traffic in real-time showing clearly the achievable packet rate. The maximum packet rate that can be handled without packet loss is 270 kpackets/s. We aim to overcome this limitation by adding SR-IOV and VMs to enable higher packet rates without packet loss.

Another important metric, as shown in our motivating example in Figure 1, is latency, especially tail-latency. We present an HDR diagram with the latencies of all analyzed traffic rates in Figure 1 showing that the differences between packet rates are significant, ranging in worst case from 688 µs until 4 s. The latencies of 50 kpackets/s are showing a mean of 68 µs rising towards more than 303 µs in worst-case with a standard deviation (STD) of 18.7 µs and a worst-case jitter of 44 µs showing low-latencies before packet loss occurs with a high jitter derivation. Lastly, the packet rates higher than 270 kpackets/s with significant packet loss have a tail latency above 70 ms due to memory overhead. Resulting in general latency fluctuations before packet loss occurs, latencies with rates causing packet loss, and, therefore, changed network behavior are behaving in general as expected. In summary, when deterministic latency and high throughput are required, we showed that another rapid prototyping and CI tool is needed compared to traditional Mininet.

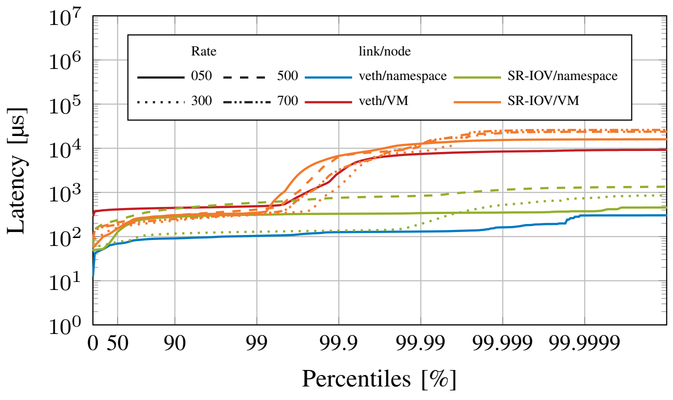


Figure 5: Latency HDR diagram for rates without packet loss for all node-link variants.

C. Throughput

One of the targets of this paper is to highlight the differences towards deterministic and high bandwidth utilization between the original and extended Mininet version. The previous subsection shows that the original Mininet version has a bandwidth bottleneck at 270 kpackets/s. We perform the same comparisons for all other variants to identify their bottleneck bandwidth and throughput possibilities. The received rate after each variant when sending 700 kpackets/s from the LoadGen is shown in Figure 4 compared to the original Mininet variant in blue.

When changing the node type from namespaces towards VMs while still utilizing veths as links, the maximum packet rate achievable decreases dramatically to a maximum of 145 kpackets/s instead of 270 kpackets/s. This decreasing maximum loss-less rate is resulting from the additional added memory copies and overhead due to adding a virtual interface to a VM instead of utilizing direct memory access (DMA) and IOMMU translations as possible when using a PCI device instead. Based on these results only changing the node type as done in Containernet 2.0 [18] is not achieving our target of deterministic and high throughput.

As we added in our approach one additional change opportunity, we analyze the impact of adapting the link type from veth to SR-IOV while using namespaces as node types to make it comparable to the original Mininet. Changing the link type from a memory copy-based virtual interface to a hardware-based interface achieves a high increase of stable throughput. With this combination we can achieve a stable rate of 680 kpackets/s in our measurements, which is an increase of 410 kpackets/s. This increased capabilities clearly show that veths as Linux implementation of virtual interfaces used in Mininet are providing the highest bottleneck due to both, the nodes and the traffic on the link, requiring capacity from CPU, memory, and system buses.

When utilizing both VMs as nodes and SR-IOV VFs as link endpoints with DMA and direct passthrough, we can achieve a even higher throughput without packet loss at 720 kpackets/s resulting due to the higher isolation of VMs compared to namespaces. This clearly shows that for experiments requiring high and deterministic as well as stable throughput capabilities, changing the link type to a hardware supported, less CPU and memory intense interface is necessary. Adding the additional overhead and higher resource demand of VMs in comparison is improving the results slightly, but the resource consumption is increasing dramatically in comparison.

D. Latency

After analyzing the maximum possible throughput without packet loss on each variant, we want to see if we can achieve a stable and low latency on each variant analyzed in this paper. Figure 1 is showing the baseline for these measurements. We consider for our further analysis only those rates without packet loss for each variant as latencies in overload scenarios are not deterministic as shown by Gallenmüller et al. [34]. Figure 5 is presenting one HDR diagram containing the latencies for all four variants for rates without packet loss. When analyzing all variants at 50 kpackets/s, we see a tail-latency for the original Mininet at 303 µs (mean 68 µs, STD 18.7 µs, worst-case jitter 44 µs), VM with namespaces at 9211 µs (mean 420 µs, STD 173 µs, worst-case jitter 60 µs), VM with SR-IOV 1596 µs (mean 196 µs, STD 371 µs, worst-case jitter 37 µs), and namespaces with SR-IOV 455 µs (mean 148 µs, STD 93 µs, worst-case jitter 37 µs) showing that in the case of no packet loss or memory overload, the original Mininet shows a nearly stable behavior with minor adaptions based on the lowest overhead due to direct memory copies and light-weight isolation, whereas both variants with SR-IOV are showing an improved jitter behavior over seven nodes. For higher packet rates, we can only analyses the two variants with SR-IOV based links as the variants with veths are causing packet loss with rates higher than 270 kpackets/s. The raise in tail-latency between the 99^{th} and the 99.9^{th} percentile for VMs is similar to results from previous works such as Wiedner et al. [26] when using partly optimized VMs. In this case, the VMs are not pinned to specific cores, and the corresponding cores are not isolated.

For the highest packet rate supporting both namespaces and VMs in combination with SR-IOV (500 kpackets/s), the

ITC Library

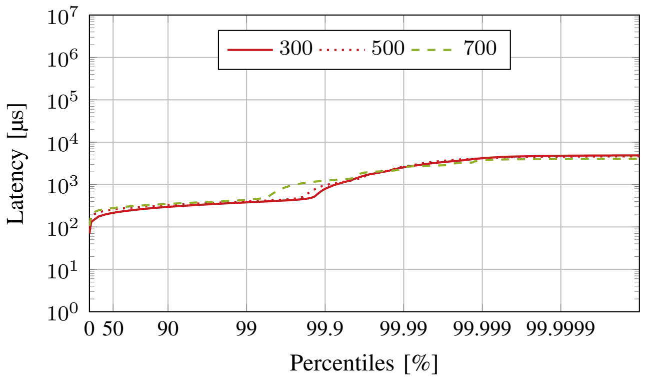


Figure 6: Latency HDR diagram on selected rates (kpackets/s) with SR-IOV VFs and optimized VMs

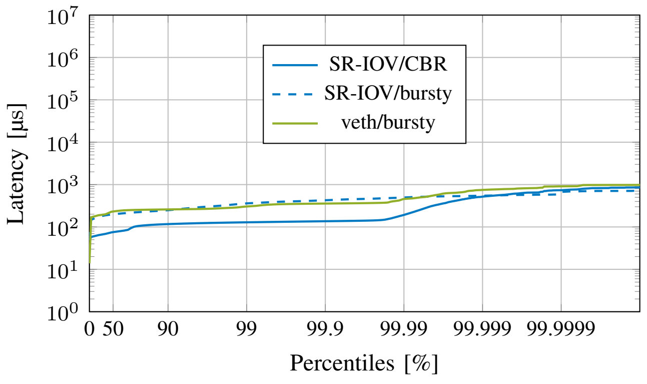


Figure 7: Latency HDR diagram comparing CBR traffic vs. bursts of 100 packets with namespaces—Rate 300 kpackets/s for CBR and traffic with bursts

latency of the namespace is rising towards 1399 µs (mean 271 µs, STD 108 µs, worst-case jitter 19 µs) and for VMs towards 23 399 µs (mean 228 µs, STD 341 µs, worst-case jitter 741 µs). The reduced jitter and similar stability in latency are caused by the change from interrupt to polling-based mode in the Linux NAPI driver. As expected, the latency is rising for both variants but showing that we can achieve a latency distribution similar to previous works such as [11]. This shows that when higher packet rates and stable latency are required, namespaces with SR-IOV are the best choice, whereas VMs require further optimizations to retrieve similar latency ranges.

E. Optimizations

Optimizing the VMs additional with pinning the virtual cores of the VM to physical cores of the hypervisor on the NUMA node the interface providing the VFs for this node is attached to. Moreover, we isolate those cores from the host kernel to reduce the impact of interrupts onto forwarding latency [27]. The downside of pinning VMs to host cores is that no load-balancing or over-provisioning of the specified cores is possible resulting in increased resource consumption and decreased flexibility. When analyzing the optimized VM in comparison to the non-optimized VM with using veth interfaces, the optimization has no influence on the maximum achievable packet rate due to the needed kernel operations which are moved to other cores. Furthermore, the maximum achievable throughput using SR-IOV links on optimized VMs in our seven-node scenario is 850 kpackets/s compared to 720 kpackets/s without optimizations. This improvement means optimizing the VMs and isolating the cores is important to achieve high throughput with Mininet, VM nodes, and SR-IOV-based links if needed.

The improvement in tail-latency shown in Figure 6 for the three highest measured and achievable rates is similar to the increase of the maximum achievable loss-less packet rate. For 500 kpackets/s are we achieving a worst-case latency of 4667 µs (mean 262 µs, STD 71 µs, worst-case jitter 14 µs) which are higher latencies than the variant with namespaces due to additional overhead but the lowest statistical derivations showing stable behavior of the latency and similar behavior for all rates. In our discussion on VM optimizations, we demonstrated through our evaluations that utilizing SR-IOV or hardware interfaces instead of veth in Mininet enhances both throughput and latency performance. While VMs generally do not introduce significant overhead, they might not yield the highest revenue. Therefore, it is advisable to use this resource overhead when an entire operating system is necessary.

F. Bursty Traffic

Moreover, in addition to our analysis of CBR traffic, we analyzed the behavior of the different variants with bursty traffic ranging between 50 to 200 packets per burst. We want to analyze whether the previous results hold with bursty traffic. We have analyzed for namespaces the latency for 300 kpackets/s in Figure 7 in comparison to SR-IOV with CBR traffic. Due to the processing of packets in a burst, this rate does not cause packet loss for the original Mininet compared to CBR traffic before. For SR-IOV with namespaces is the tail-latency at 1040 µs with bursty traffic which is slightly lower than for CBR traffic, whereas as expected the mean rises significantly to 223 µs (STD 36 µs, worst-case jitter 59 µs) as shown in Figure 7.

In Figure 8 is the same showing for VMs with a rate of 700 kpackets/s comparing between non-optimized and optimized VM with bursty and CBR traffic. For the non-optimized VM, slight variations are visible in a small range. In contrast, the optimized VM differs only in that the jump to higher latencies after the 99.9^{th} percentile is slightly higher for bursty traffic. This analysis shows that with our extended Mininet, we are able to handle bursty traffic at a high throughput rate, especially when using SR-IOV in combination with optimized VMs or namespaces.

VII. Limitations

In addition to the benefits we show in our measurements, the results are limited to simple packet processing and a simple topology. Adding computations to the packet processing

TTC Library

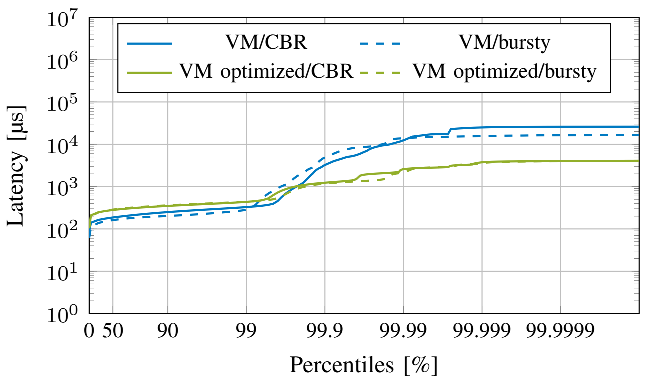


Figure 8: Latency HDR diagram comparing CBR traffic vs. bursts of 100 packets with VMs—Rate 700 kpackets/s for CBR and traffic with bursts

reduces the available CPU time and increases the latency, whereas the throughput might not even be affected. We can interpret the analyzed highest achievable packet rate as upper bound of potential achievable packet rates. Similar, our analyzed simple line topology shows the performance differences, which can widely differ in more complex scenarios, but the simple topology already shows the potential benefits for more complex topologies due to significantly lower latencies.

Moreover, our measurement setup is used to reduce the measurement impact of an external load generator to retrieve high and constant bit rates. Due to limited time and space, we leave an analysis of the impact of packet generation within Mininet on our results for future work.

Furthermore, using SR-IOV to provide links in Mininet limits the number of available links to the number of VFs the used NIC allows. The hardware enforces this limitation due to the limited number of registers for the different separated features of VFs with the same PF. This number can be increased by using multiple pairs of NICs.

VIII. Recommendations

Depending on the specific requirements of the experiment, CI, or DTN conducted with our extended version of Mininet, one of the four variants illustrated in Figure 2 can be chosen.

If resource constraints are a critical concern and neither throughput nor latency needs to be prioritized in the measurements, we recommend using the original Mininet or our extended variant that employs namespaces and veth-based links. These options have the lowest impact on resources, though they come with a slight trade-off in performance regarding packet loss.

In scenarios where an entire operating system is necessary on each node, and throughput or latency considerations are not applicable, the best approach is to utilize VMs as nodes along with veth links. While this option may result in the worst overall performance, it does not require additional NIC and mainboard support.

Table II: Summarized Recommendations

Technology	Latency	Throughput	Resources
Namespace			
veth	✗	✗✗	✓✓
SR-IOV	✓✓	✓	✓
VM			
veth	✗✗	✗✗	✗
SR-IOV	✓	✓✓	✗✗

When an entire operating system is needed on each node along with high throughput or low latency, using VMs in combination with SR-IOV on Mininet can offer significant benefits. If resources allow, we recommend dedicating time to optimize both the VMs and the hypervisor, as this can lead to significantly improved performance.

For all other situations, we suggest using namespaces combined with SR-IOV-based links. This configuration provides low and stable latency, as well as a significant enhancement in the maximum packet rate achievable with zero packet loss.

We summarize our recommendations in Table II.

IX. Reproducibility

We published the extended Mininet version as CIMininet[1]. Furthermore all instructions and scripts to reproduce our results, additional measurements and raw data are available on the accompanying website[2]. The raw data are available on mediaTUM[3].

X. Conclusion and Future Work

Continuous Integration and Digital Twins are current and future technologies enabling the automation of deployment and configuration processes. As networks are complex and multiple devices are involved, realistically, emulating networks to use them in CI or DTNs is challenging. This paper analyzes the possibilities of extending the network emulator Mininet to provide high, deterministic usable bandwidth and deterministic, low tail latency. Mininet uses Linux namespaces to isolate different nodes from each other and veths to provide virtual links on the current node. Due to shared usage of memory, system buses, and CPU, multiple potential non-network-induced bottlenecks result.

By extending Mininet to support VMs as nodes and SR-IOV-based links, we show that significant improvement of both bandwidth and latency is possible simultaneously. With our measurements on a seven-node line topology using an external timestamper and load generator, we show that using hardware-supported interfaces and cabled connections improves the highest zero loss packet rate from 270 kpackets/s to 850 kpackets/s. It provides as well a significantly lower and more stable latencies. To conclude, using our prototype to verify and test latency-sensitive or high throughput applications, services, or protocols is possible using the standard

[1]https://github.com/tumi8/CIMininet
[2]https://tumi8.github.io/mininet-vm-sriov
[3]https://doi.org/10.14459/2025mp1773238

ITC Library

Mininet API. This allows testing scripts and automating the integration of network applications in CI processes to improve the quality of software, services, and tools in networks.

Further, we plan to analyze the influence of complex scenarios, computation-heavy applications, and Docker containers as nodes together with SR-IOV in the future to improve the possibilities of network emulation for latency and throughput-sensitive applications. We further plan to use our published data to provide over time modeling and analyze our data's predictability.

Acknowledgments

This work was supported by the EU's Horizon 2020 programme as part of the projects SLICES-PP (10107977) and GreenDIGIT (4101131207), by the German Federal Ministry of Education and Research (BMBF) under the projects 6G-life (16KISK002) and 6G-ANNA (16KISK107), and by the German Research Foundation (HyperNIC, CA595/13-1).

References

[1] M. Fowler and M. Foemmel, "Continuous integration," 2006.

[2] Y. Wu, K. Zhang, and Y. Zhang, "Digital twin networks: A survey," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13 789–13 804, 2021.

[3] P. Almasan, M. Ferriol-Galmes, J. Paillisse, and J. Suárez-Varela, "Digital Twin Network: Opportunities and Challenges," Jan 2022. [Online]. Available: https://arxiv.org/pdf/2201.01144

[4] A. Mozo, A. Karamchandani, S. Gómez-Canaval, M. Sanz, J. I. Moreno, and A. Pastor, "B5gemini: AI-Driven Network Digital Twin," *Sensors*, vol. 22, no. 11, p. 4106, 2022.

[5] H. Wang, Y. Wu, G. Min, and W. Miao, "A graph neural network-based digital twin for network slicing management," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 2, p. 1367–1376, 2022.

[6] M. Saravanan, P. S. Kumar, and A. R. Kumar, "Enabling network digital twin to improve QoS performance in communication networks," *IEEE Transactions on Industrial Informatics*, 2022.

[7] G. Electric, "The Digital Twin: Compressing Time to Value for Digital Industrial Companies," *General Electric*, 2018.

[8] R. Söderberg, K. Wärmefjord, J. S. Carlson, and L. Lindkvist, "Toward a Digital Twin for real-time geometry assurance in individualized production," *CIRP annals*, vol. 66, no. 1, pp. 137–140, 2017.

[9] U. Muslim and S. Recker, "Emulation Platform to Build Digital Twins of Edge Computing Environments," in *2024 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2024, pp. 512–514.

[10] Y. Zheng, S. Yang, and H. Cheng, "An application framework of digital twin and its case study," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, pp. 1141–1153, 2019.

[11] F. Wiedner, M. Helm, S. Gallenmüller, and G. Carle, "HVNet: Hardware-Assisted Virtual Networking on a Single Physical Host," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops, INFOCOM 2022 - Workshops, New York, NY, USA, May 2-5, 2022*. IEEE, 2022, pp. 1–6.

[12] R. M. Fujimoto, K. S. Perumalla, and G. F. Riley, *Network simulation*. Springer Nature, 2022.

[13] A. Varga, "OMNeT++," in *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.

[14] A. K. Saluja, S. A. Dargad, and K. Mistry, "A Detailed Analogy of Network Simulators—NS1, NS2, NS3 and NS4," *Int. J. Future Revolut. Comput. Sci. Commun. Eng*, vol. 3, pp. 291–295, 2017.

[15] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International conference on communication, computing & systems (ICCCS)*. IEEE, 2014, pp. 139–42.

[16] J. Yan and D. Jin, "Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015, pp. 1–7.

[17] M. Peuster, H. Karl, and S. van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 148–153.

[18] M. Peuster, J. Kampmeyer, and H. Karl, "Containernet 2.0: A Rapid Prototyping Platform for Hybrid Service Function Chains," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 335–337.

[19] Y. Dong, X. Yang, X. Li, J. Li, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," in *HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, 2010, pp. 1–10.

[20] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support," in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, Apr. 2010, pp. 1–12.

[21] Z. Tang, D. Chen, T. Sun, L. Zhang, M. Qi, and X. Wang, "Intelligent Awareness of Delay-Sensitive Internet Traffic in Digital Twin Network," *IEEE Journal of Radio Frequency Identification*, vol. 6, pp. 891–895, 2022.

[22] E. Soares, G. Sizilio, J. Santos, D. A. Da Costa, and U. Kulesza, "The effects of continuous integration on software development: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 3, p. 78, 2022.

[23] M. Vucnik, T. Solc, U. Gregorc, A. Hrovat, K. Bregar, M. Smolnikar, M. Mohorcic, and C. Fortuna, "Continuous Integration in Wireless Technology Development," *IEEE Communications Magazine*, vol. 56, no. 12, pp. 74–81, 2018.

[24] S. Rieger, L.-N. Lux, J. Schmitt, and M. Stiemerling, "DigSiNet: Using Multiple Digital Twins to Provide Rhythmic Network Consistency," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–5.

[25] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, "Performance Overhead Comparison between Hypervisor and Container Based Virtualization," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, 2017, pp. 955–962.

[26] F. Wiedner, M. Helm, A. Daichendt, J. Andre, and G. Carle, "Performance evaluation of containers for low-latency packet processing in virtualized network environments," *Perform. Eval.*, vol. 166, no. C, Jan. 2025.

[27] S. Gallenmüller, J. Naab, I. Adam, and G. Carle, "5G QoS: Impact of Security Functions on Latency," in *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*. IEEE, 2020, pp. 1–9.

[28] T. Zhang, L. Linguaglossa, J. Roberts, L. Iannone, M. Gallo, and P. Giaccone, "A benchmarking methodology for evaluating software switch performance for NFV," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, Jun. 2019, pp. 251–253.

[29] G. K. Lockwood, M. Tatineni, and R. Wagner, "SR-IOV: Performance Benefits for Virtualized Interconnects," in *Annual Conference of the Extreme Science and Engineering Discovery Environment, XSEDE '14, Atlanta, GA, USA - July 13 - 18, 2014*, S. A. Lathrop and J. Alameda, Eds. ACM, 2014, pp. 47:1–47:7.

[30] A. Daichendt, F. Wiedner, J. Andre, and G. Carle, "Applicability of Hardware-Supported Containers in Low-Latency Networking," in *2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, 2024, pp. 1–7.

[31] AMD, "Performance Tuning Guidelines for Low Latency Response on AMD EPYC-Based Servers Application Note," Jun. 2018, URL: https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/tuning-guides/56263-EPYC-performance-tuning-app-note.pdf, Last accessed: Jan 10, 2025.

[32] Intel Corporation, "E810 datasheet rev2.5," URL: https://cdrdv2-public.intel.com/613875/613875_E810_Datasheet_Rev2.5.pdf, Last Accessed: Jan 10, 2025.

[33] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, K. Cho, K. Fukuda, V. S. Pai, and N. Spring, Eds. ACM, 2015, pp. 275–287.

[34] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "Ducked tails: trimming the tail latency of (f) packet processing systems," in *2021 17th International Conference on Network and Service Management (CNSM)*. IEEE, 2021, pp. 537–543.

TTC Library