On Improving Parallel Real-Time Network Simulation for Hybrid Experimentation of Software Defined Networks

Mohammad Abu Obaida
School of Computing and Information Sciences
Florida International University
Miami, Florida 33199
mobai001@fiu.edu

Jason Liu
School of Computing and Information Sciences
Florida International University
Miami, Florida 33199
liux@cis.fiu.edu

ABSTRACT

Real-time network simulation enables simulation to operate in real time, and in doing so allows experiments with simulated, emulated, and real network components acting in concert to test novel network applications or protocols. Real-time simulation can also run in parallel for large-scale network scenarios, in which case network traffic is represented as simulation events passed as messages to remote simulation instances running on different machines. We note that substantial overhead exists in parallel real-time simulation to support synchronization and communication among distributed instances, which can significantly limit the performance and scalability of the hybrid approach. To overcome these challenges, we propose several techniques for improving the performance of parallel real-time simulation, by eliminating parallel synchronization and reducing communication overhead. Our experiments show that the proposed techniques can indeed improve the overall performance. In a use case, we demonstrate that our hybrid technique can be readily integrated for studies of software-defined networks.

CCS CONCEPTS

• **Networks** → **Network simulations**; **Network experimentation**; • **Computing methodologies** → **Simulation environments**; **Simulation tools**; **Parallel computing methodologies**;

KEYWORDS

Real-time network simulation, parallel simulation, emulation, software-defined networking, SDN

ACM Reference format:
Mohammad Abu Obaida and Jason Liu. 2017. On Improving Parallel Real-Time Network Simulation for Hybrid Experimentation of Software Defined Networks. In *Proceedings of 10th EAI International Conference on Simulation Tools and Techniques, Shenzhen, China, September 2017 (SIMUtools'17),* 9 pages.
https://doi.org/10.475/123_4

SIMUtools'17, September 2017, Shenzhen, China

ACM ISBN 123-4567-24-567/08/06...$15.00
https://doi.org/10.475/123_4

1 INTRODUCTION

Network Innovations can be disruptive. A case in point is the design of Future Internet Architectures (FIA). Recent proposals on Software Defined Networking (SDN) have gained tremendous momentum in the networking community, both in data-center networks and enterprise networks, due to SDN's advantages in flexibility, programmability and functionality over the traditional network design. With the growing adoption of SDN-enabled hardware and protocols, one needs network testbeds that can evaluate SDN-based solutions and help understand at-scale application behaviors under diverse network conditions before real deployment.

Various testbeds have been proposed to evaluate network solutions with different capabilities. Emulation testbeds ensure a faithful representation of the operations of the relevant systems and protocols. For example, Mininet [11] is an emulation tool that has been widely used for prototyping and validating research ideas for SDN and cloud services. A major concern related to emulation, however, is its limitations in performance, especially under stress situations where network traffic exceeds the processing capability of the emulation software. It is well-known that the aggregate throughput handled by Mininet during an experiment must not exceed several gigabits per second, or significant errors may occur that one would not be able to trust the performance results from the experiments. Emulation also has a scalability issue: it is rather difficult to emulate large-scale networks (with tens of thousands of networked entities or more) in practice.

Simulation provides a good alternative. It is especially useful for comparing design alternatives and for understanding the performance issues and scalability limitations under different scenarios. Simulation is also regarded as a good way to prototype network protocols, so that their major functions can be quickly examined in order to identify potential issues with the design. A major problem with simulation, however, is that it lacks realism, especially when one wants to study new network designs and new network protocols under real network operating environments. Furthermore, in the most common cases, the implementation effort for simulation is not much less than implementing the real software because of the complexity involved in the today's network simulation tools. Consequently, simulation can seem to be be less attractive to researchers who demand more effective and fast implementation of their research ideas. As a result, most of the design evaluations and performance studies today are

concerned only with the algorithmic and protocol-level details, but ignore the network-wide issues. By treating the large network as a black box, these studies may not reflect the reality. For example, how well will the new network design scale and accommodate various types of traffic when deployed? How would an SDN-enabled transport protocol or a content distribution service potentially alter network traffic characteristics at large and therefore affect existing applications?

Real-time simulations has been used to improve the utility of simulation for network research (e.g., [1, 5, 17, 19, 21, 24]). By combining the simulated network with emulated components, such as running real implementation of applications and protocols in virtual machines and having simulation to interoperate with software network switches and routers, real-time simulation provides as a hybrid testing environment for studying network protocols and applications in large-scale settings and under various synthetic network conditions.

Parallel real-time simulation extends parallel simulation tools with real-time capabilities [19]. In doing so, one can improve the scalability of an emulated network when the simulated network entities and simulated network traffic are combined with the emulated network protocols and applications implemented for the real system. While parallel real-time simulation potentially can support large-scale experiments, the communication overhead for synchronizing the parallel simulation processes can be substantial especially for large parallel runs. In addition, since the network traffic traveling across separate simulation instances is represented as remote time-stamped messages between the parallel processes, if the traffic demand increases, the cross-machine traffic for the simulation will also increase. For traffic intensive applications, the traffic volume can overwhelm the system causing significant slowdowns in simulation. When the real-time execution can no longer be guaranteed, we cannot trust the real-time simulation results as they no longer reflect the real operation of the system.

In this paper, we propose several methods to improve parallel real-time network simulation. First, we propose a drastically different design of parallel simulation. Given that most of today's parallel architectures are tightly synchronized in wall-clock time, we removed the conservative synchronization protocol that help guarantee causality constraint for parallel simulation. Instead, the simulation instances advance their simulation time in coordination with the local wall-clock time. In doing so, the real-time simulation system may introduce subtle timing errors caused by out-of-order event execution. We posit that these errors are relatively minor when compared to the potential errors otherwise caused by the slowdown of the simulation processes due to fine-grained synchronization.

Second, we introduce methods and the corresponding software constructs to further reduce the communication overhead when simulation is conducting traffic on behalf of the emulated applications. This is achieved, for example, by batching and compressing the real network packets as they are transported across the machine boundaries.

The rest of this paper is organized as follows. Section 2 provides the background of our research. We discuss related work in parallel real-time simulation and in SDN emulation. Section 3 presents the overall design of our new parallel real-time simulation framework. Section 4 focuses specifically on the design of the software constructs in real-time simulation to interact with the emulated networks and applications. We conducted preliminary experiments to evaluate our proposed approach. The results are presented in Section 5. Finally, we conclude the paper and outline future work in Section 6.

2 BACKGROUND

In this section we discuss the state of the art on parallel real-time simulation and also related work supporting large-scale SDN experiments.

2.1 Parallel Real-Time Simulation

Parallel discrete-event simulation (PDES), or parallel simulation, is well studied field. A PDES system consists of separate simulator instances that require synchronization in order to guarantee the natural advance of simulation time at various components of the system. This is achieved by exchanging time-stamped message among the parallel instances. There are two broad categories of synchronization algorithms, optimistic and conservative. A comprehensive review of these approaches can be found in [6]. Parallel simulation has been used to model large-scale systems, such as transportation systems, computer networks, parallel systems and parallel applications. Since these systems typically involve tightly coupled components, efficient time management is one of the most important factor in determining the performance and scalability of a parallel simulator.

When the simulation time advancement is synchronized with the wall-clock time, the state of the target system is changed in real time. That is, the simulation is running in real time. If a simulated network is running in real time, it is able to interact with the physical processes that happen in real time. That will include interacting with the users in real time, and interacting with the real applications and network protocols operating in real time [12]. By interaction, we mean that a real entity (a user, an application, or a network protocol) can interoperate with an simulated one. For example, a simulated TCP protocol implemented with a congestion control algorithm can exchange data packets with a real TCP protocol implemented in the operating system on a real machine. In doing so, the simulated network can be treated as an emulated component and therefore be able to test applications or protocols, and play with simulated network scenarios.

Many existing network simulators have been augmented to support real-time simulation, such as ns-2 [5] and ns-3 [20]. To improve simulation performance, one can adopt parallel discrete-event simulation. Previously, we developed PRIME,

which is a discrete-event simulator designed to run on parallel and distributed platforms and handle large-scale network models [19]. PRIME has an emulation infrastructure based on VPN that allows distributed client machines to remotely connect to the real-time simulator running at a high-performance computing cluster [15]. PRIME also provides implementations of a set of TCP congestion control algorithms, ported from the Linux TCP implementation, so that network applications can readily interoperate with the parallel real-time simulator for extensive simulation and emulation studies [4]. PRIME also incorporated other advanced network traffic modeling techniques. For example, PRIME supports integration of fluid-based network traffic with packet-oriented network traffic so that the simulator can handle large traffic volume with reduced computation [14].

To run parallel simulation in real time, separate simulation instances are run concurrently on distributed-memory machines. Each simulation instance is operated in real time. That is, its simulation events are processed according to the wall-clock time. The simulation instances communicate with one another using time-stamped messages; their time advancement is synchronized according to the (conservative) parallel simulation synchronization protocol. In [13], we proposed a solution for scheduling the logical processes according to the deadline and the real-time constraints. We also proposed an event delivery mechanism for the parallel simulator to incorporate emulated events originated from the physical system.

Parallel real-time simulation allows hybrid experiments of large networks. A network experiment in this case consists of a simulated network with a detailed specification of the network topology, with end hosts, routers and links, with detailed parameters, including bandwidth and latency. The experiment also specifies network protocols and applications running on the hosts and routers. Some part of the network can be designated as real, in which case, real implementations of network applications and network protocols, and real network traffic can interact with the simulated network. This would allow us to study them directly in their real machine environments. Using parallel real-time simulation, we can test these applications and protocols by embedding them in a large-scale virtual network environment, so that we can create diverse network conditions in simulation for validation and performance evaluation.

As an example, Figure 1 shows a parallel real-time simulation setup of a hybrid experiment involving a large simulated network. The large simulated network is partitioned among N simulation instances and each assigned to a separate machine (we call a worker). There can be one or several emulated networks connected to each simulation instance. In this example, we have an emulated SDN network (which we call SDN SimpleNet) for each simulation instance. The SDN SimpleNet is an emulated network, composed of Linux namespaces, software routers [18], and one or more controllers for managing the OpenFlow network [16]. They can interact with the simulated network through TCP/IP. Packets generated by the

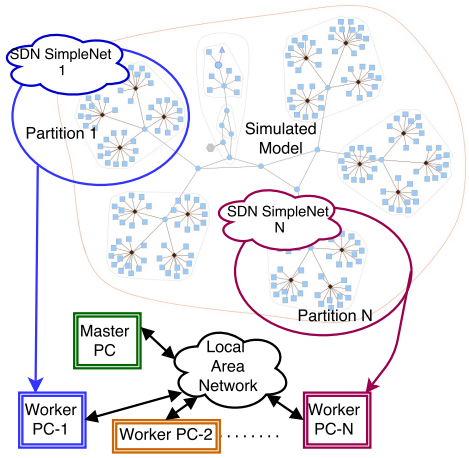

Figure 1: Parallel real-time simulation of a distributed SDN solution on a large hybrid network.

emulated network, if they need to be sent over the simulated network, will be converted to simulated events. If the destination of the packets is located in another simulation instance, the machine running the local simulation instance will send these events to the remote instances through its event delivery mechanism. The events will be processed at the remote instance according to their timestamps. If the packets are destined to an emulated network (another SDN SimpleNet), the simulator will export the events from the simulation and give the packets to the target emulated network (say, using raw sockets).

2.2 SDN Emulation

Over last decade there has been tremendous push towards SDN. OpenFlow[16] has emerged as a framework to standardize the communication between the control plane and the data plane of an SDN network. Software-Defined eXchange (SDX) (such as [7]) is also emerging as an standard for information exchange across domains. Aside from these standardization efforts, Open vSwitch [18] and Linux namespace has enabled OpenFlow-based SDN prototyping. One of the most widely used SDN prototyping emulator is Mininet ([8, 11]) which can create a complete virtual network on a single machine. The virtual network consists of lightweight hosts implemented as separate Linux namespaces, which are connected by software switches controlled by SDN controllers.

As a widely used platform, Mininet helps prototype and perform basic tests of SDN applications. The bulk of Mininet implementation is related to providing a python-library that glue together several lightweight Linux-specific virtualization components, virtual switches and flow control mechanism in a network experiment. One can create a few thousand of hosts using Mininet easily; however, trying to invoke a large number of traffic flows would cause the emulator to perform very poorly and result in significantly low throughput [10].

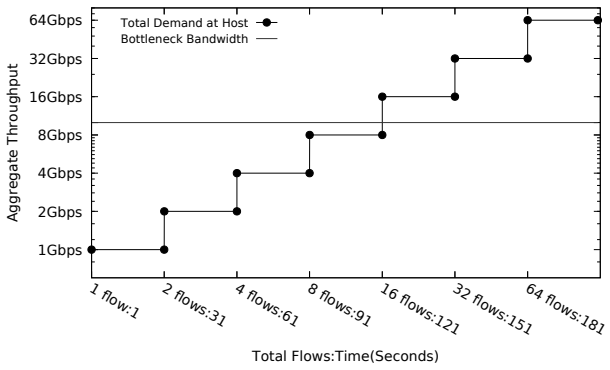

Figure 2: Traffic demand over time.

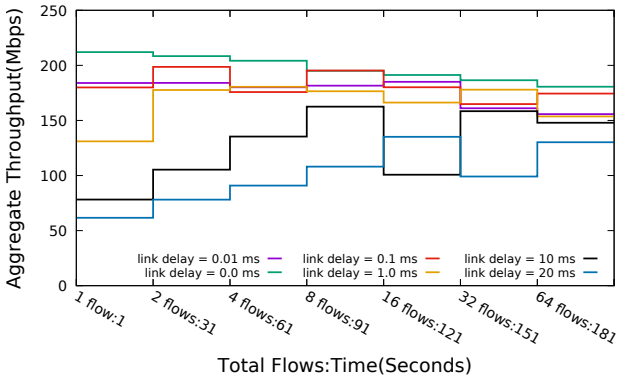

Figure 3: Aggregate throughput by Maxinet.

An interesting aspect of Mininet is that it can be used to support reproducible networking experiment research. This is achieved by Mininet allowing research projects to recreate the same network experiments in one machine with very little effort [8, 9]. Some of these reproducible projects tackle SDN research, like Hedera [2], which is an Openflow-enabled load balancing solution. However, the downside of Mininet is that some research requires an extensive performance evaluation part that could be difficult to reproduce. Thus, some of the techniques used to reproduce Hedera[1] include downscaling the link bandwidth in the setup from 1 Gbps to 10 Mbps and also restrict the CPU utilization of each host to a tiny fraction of the total CPU to prevent traffic generation and I/O contention. Because of some of these design decisions, whether we can truly reproduce research results is questionable.

In spite of wide spread acceptance, Mininet has some severe performance and scalability issues. More specifically, its performance may fall short when it comes to creating large networks consisting of hundreds of hosts and running series applications. To solve this problem, Yan et al. proposed virtual-time-enabled VT-Mininet, a technique intended to improve performance fidelity [23]. They implemented virtual clock in the Linux kernel so that Mininet can experience time dilation to improve its packet processing capability. However, Wette et al. [22] showed that the relationship between the required dilation factor and the size of the topology is not linear for larger networks, which means using time dilation is not sufficient to supporting large experiments.

To solve the network size limitation of Mininet, Maxinet [22] was proposed to partition the network into smaller pieces and distribute them among separate Mininet instances running currently on a cluster. A disadvantage of using Maxinet is the cross-machine traffic, which is still limited by the bandwidth of the physical network connecting the partitions.

To illustrate such limitations, we created a distributed Maxinet experiment. We set up a network with a tree topology, which consists of 128 hosts partitioned into two equal parts and mapped onto two physical machines. We varied the number of flows from 1 to 64, doubling number of flows every 30 seconds, with at most one flow at each host. We direct the flows from one partition to the other. The experiment was run on two 16-core machines connected with a 10 Gbps network on CloudLab [3]. Figure 2 shows the traffic demand of this experiment (i.e., the theoretical limit). Results of the experiment are shown in Figure 3. We can see that the aggregate throughput never exceeds 220 Mbps, which is significantly lower than the demand. In the experiment, we also tried to vary the link delay in order to control the TCP throughput. However, the severe limitation in the data throughput across the machines makes these choices almost irrelevant.

[1] https://reproducingnetworkresearch.wordpress.com/2012/06/06/hedera/

3 OVERALL DESIGN

The overall architecture of our parallel real-time simulation testbed is comprised of one or multiple real-time simulation instances each interacting with an arbitrary number of emulated networks. Each of the real-time simulation instance is a stand-alone simulation execution with support for real-time interaction with the emulated networks. We use the real-time simulator based on PRIME [19]. PRIME provides packet capture and forwarding capability implemented at the simulated network interfaces. The packet capture and forwarding mechanism is called a *portal* in the simulator. The simulator can instantiate a large number of portals at the same time, allowing it to interact with separate emulated networks. For SDN experiments, these emulation networks are called SDN SimpleNets. They are lightweight implementation of Mininet-like networks, which consist of virtual hosts, implemented as Linux namespaces, and software routers, implemented using Open vSwitch.

The real-time simulation instances coordinate with one another through messages delivered in real time. Each simulation instance advances its simulation clock according to the local wall-clock time. We assume that the machines running the real-time simulators are themselves time synchronized, for example, using the Network Time Protocol (NTP). In a cluster or cloud computing environment, this is generally true.

Communication between the real-time simulation instances can take place either when a simulation event needs to be sent to a remote simulation instance (for example, between a pair of simulated TCP end-points located on separate real-time

simulation instances), or when an emulated network needs to communication with another emulated network attached to a different real-time simulation instance. We treat communication crossing the boundaries of real-time simulation instances as network packets and serialize necessary data into a raw buffer and send them either through a TCP connection established between the real-time simulation instances or as individual UDP packets. The receiver side can then perform required translation upon the arrival of the packets based on the information contained in them. We call this mechanism *packet marshalling*.

In a traditional parallel real-time simulation, the real-time simulation instances are tightly synchronized using a parallel simulation synchronization protocol. Depending on the lookahead obtained when partitioning the target network model (which usually ranges from a few microseconds to possibly several milliseconds), the cost of synchronization can be quite significant, especially when the number of parallel instances increases. The synchronization cost may eventually drag down the real-time simulation performance, making it impossible to keep up with the real time.

Our proposed method is to totally relieve the real-time simulation instances from the burden of having to perform time synchronization (conservative or optimistic) at all. The simulation instances can advance their simulation time independently, although they can only advance their simulation clocks according to their local wall-clock time. Failures may occur when delivering the packets or missing real-time deadlines due to delays in event processing or network latencies. However, these failures would remain local to the specific instances, and they would not impact the progression of the entire simulation, which may consist of hundreds or thousands of these real-time instances in a large network experiment scenario. Another benefit of this approach is that, since the instances are stand-alone simulations, they can be created and destroyed as needed, and therefore one can presumably create a more dynamic simulation environment.

Figure 4 shows an example of parallel real-time simulation for hybrid SDN experimentation at-scale. A complete experiment is comprised of multiple real-time simulation instances, each of which runs on a separate machine. These instances are formed of an arbitrary number of SDN SimpleNets connecting to a real-time simulation instance, with packet capture and forwarding capabilities.

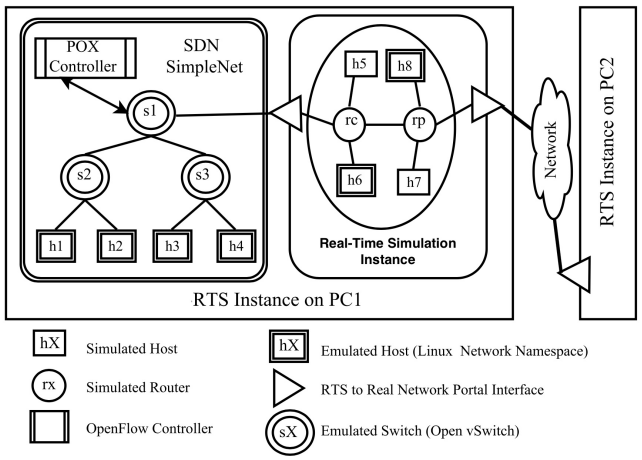

Figure 4: Independent parallel real-time simulation.

4 REAL-TIME PORTALS

The proposed parallel real-time simulation infrastructure features two different types of real-time portals. Real-time portals are mechanisms used by the real-time simulation instances to communicate with emulated networks and with other real-time simulation instances. The first type is called the emulation portals, which connect the real-time simulator with the SDN SimpleNets. The second type is called the peering portals, which connect with remote simulation instances. The emulation portals can also be used to connect simulation instances. However, as we will show from experiements, the peering portals are better optimized for such cases.

4.1 Emulation Portals

Emulation portals use existing technology. We describe them here for completeness. When a real-time portal is configured to communicate with an emulated network, it needs to capture the packets generated by the emulated network and import them to the simulator. For emulation portals, we can capture the raw Ethernet frames using options provided in the `libpcap`[2] library. We filter the captured packets for compatibility with the protocols supported by the network simulator. Then we create a simulation event that encapsulates all necessary information of the incoming packet. The event is then scheduled and the corresponding packet is associated with the portal's attached network interface. The simulation event will be processed by the simulator kernel as soon as it gets to process it. When this event is processed, the packet will be added to the tail of the network queue corresponding to the network interface.

If the packet is destined to an IP address that is mapped to another emulated network, the event needs to be sent out through the emulation portal. In this case, the simulator will serialize the internal event data structure into an IPv4 packet and forwards it using raw IP socket. Packets that are sent out using the emulation portals are full IP packets.

Since the portal is serving as a packet capturing mechanism, any alternative packet capturing mechanism would do the job. However, their performance may vary due to different handling of context switches required to implement packet arrival. When a network simulation is configured to interact with SDN emulations, all of the portals are implemented as emulation portals so that identical processing can be take place in all places.

[2] http://www.tcpdump.org/

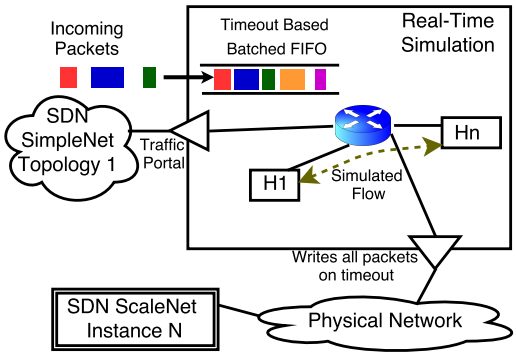

Figure 5: Receiver batching at peering portal.

4.2 Peering Portals

For communication between parallel real-time simulation instances, the simulator forwards minimal information required to reconstruct the packet event on the receiving end. In case of pure simulation (i.e., no emulated networks), this minimal information would be mainly protocol headers; in particular, no message payload is necessary. As such, these messages are usually smaller than 200 bytes. For emulated packets, the payload must be included since the emulated networks run real applications. In both cases, this information contained in the packet event needs to be contained in a real UDP packet and sent to the remote simulation instance.

To improve efficiency, one can perform batching of the packet events destined to the same remote real-time simulation instance. This is done at the sender side. By piggybacking these packet events together into one UDP packet, we can essentially amortize the cost associated with forwarding individual packets and the cost of interruption at the receiver end for processing the packet arrival. Batching can be performed either at regular time intervals, or by the packet size. Whenever there is no room left to fill in more packets in the current UDP packet, a new UDP packet should be used.

Batching can also be applied at the receiver end. Since the traffic inserted into simulator are incoming from real network interfaces, there may not exhibit any pattern. But holding the packets at the receiving portal will amortize cost of per-packet processing, given that the portal has large enough buffer and if the application can tolerate a certain level of delays. Figure 5 illustrates the receiver side batching method. The sender side logic is similar.

This effectiveness of this technique, however, requires careful calibration. On the one side, the smaller the message size compared to the maximum Ethernet frame size, the higher the benefit of using this technique. For a large number of packets, the penalties of crossing the machine boundaries can be quite high because of the overhead associated with the context switch between the user-space and the kernel. On the other hand, batching may create problems for delay sensitive applications. Even for TCP, a prolonged and unsteady batching delay may affect the calculation of the round-trip time

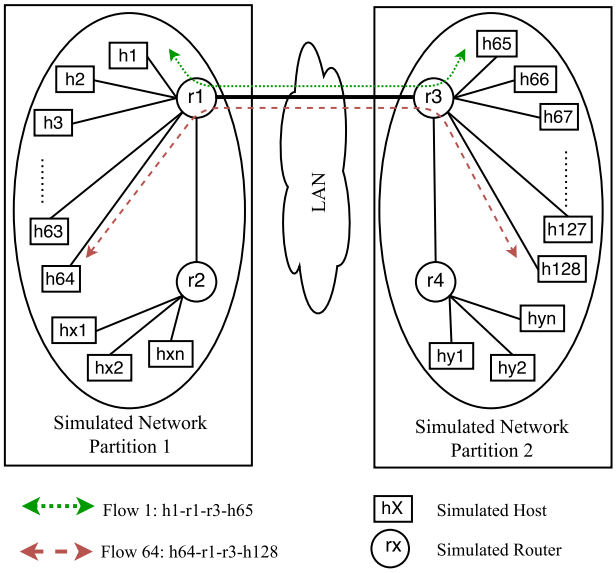

Figure 6: Dumbbell experiment setup.

and therefore affect its throughput becase TCP's congestion control algorithm may be quite sensitive to the delays. It is possible that different batching polices can be applied to different applications or different protocols, depending on their delay sensitivity. Batching technique can also be applied to achieve different objectives, such as increasing throughput, or reducing CPU utilization, etc. A careful investigation is warranted and we postpone it to future work.

5 EXPERIMENTS

We have carried out preliminary experiments to evaluate the capabilities and assess limitations of proposed parallel real-time simulation architecture. In our experiments, we were able to test networks consisting of thousands of simulated and emulated hosts and a large number of real application and simulated background traffic.

5.1 Traditional Parallel Simulation

To understand the strengths and shortcomings of the proposed parallel real-time simulation approach, we first look at the traditional approach, which is based on conservative synchronization (implemented using the Message Passing Interface). We created a simple dumbbell topology with a 10 Gbps bottleneck link, as shown in Figure 6. All the links connecting hosts (`h1` to `h128`) with the routers are configured as 1 Gbps links with 1 ms propagation delay. The network is divided into two halves and each partition is mapped to a different physical machine. That is, the bottleneck link connecting the two routers at either side of the dumbbell is a cross-machine connection between two real-time simulation instances. We start the model with one simulated flow and then double the number of flows every 30 seconds during

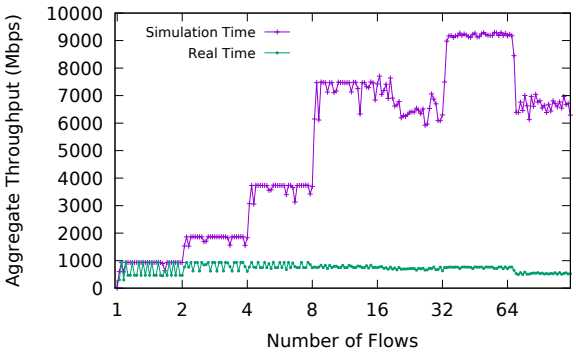

Figure 7: Parallel simulation results.

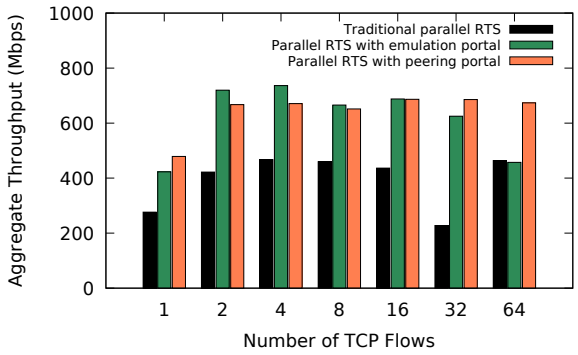

Figure 8: Real-time simulation results.

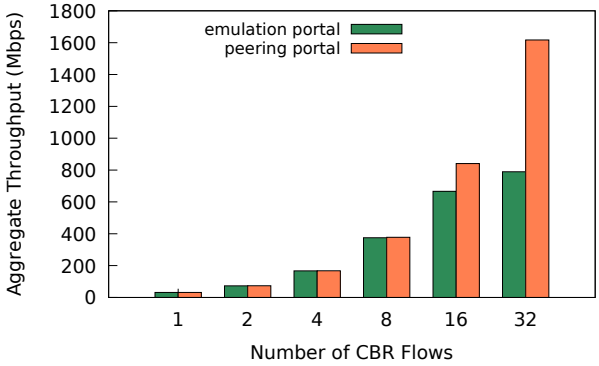

Figure 9: Results from UDP CBR traffic.

simulation until there are sixty four flows in total in the system. The source and destination of these flows lie in different partitions.

The aggregate traffic demand resembles a staircase function as depicted in Figure 2, clearly showing the demand is doubling at each successive interval. Note that the maximum aggregate bandwidth achievable in this configuration is 10 Gbps because of the bottleneck link. We conducted the experiment on two commodity workstations with eight-core Intel Xeon E5645 at 2.4 GHz clock frequency. The machines were connected by an on-board 1 Gbps network interface and a commodity 8-port switch.

Figure 7 shows the aggregate throughput of all the concurrent flows. We ran this experiment with a pure simulation setup. That is, we disabled the real-time regulation in the simulator and let the simulator run as fast as possible. The total simulation time is set to be 210 seconds, and the simulator took 1,500 wall-clock seconds to complete (a slowdown of more than 7x). Since the design bottleneck was 10 Gbps, in the presence of 16 or more flows (at 120 seconds into simulation and beyond), the throughput of the application is throttled back.

We mapped the simulation time into real time and show the aggregate throughput thus achieved by parallel simulation (i.e., the curve marked "Real Time" in Figure 7). The throughput is consistently below 1 Gpbs. There are two major factors at play that prevent parallel simulation from reaching the peak throughput in communication. First, the synchronization required by conservative parallel simulation can block the logical processes and thus slow down the event processing. Second, the parallel simulator uses MPI for sending and receiving events with remote simulation instances. Marshaling packet events unavoidably generates overheads, both in terms of the inflated message size and in terms of the processing cost for packing and unpacking the packet data.

5.2 Real-Time Experiments

To evaluate the proposed parallel real-time simulation method, we conducted another set of experiments, this time running the simulation in real time. We use the same experiment setup as shown in Figure 6. We compare three different arrangements. In the first arrangement, as the base case for comparison, we use the traditional parallel real-time simulation method, that is, running simulation in real time with conservative parallel synchronization among the multiple instances. In the second arrangement, for the peering portal implementation, we use the existent emulation portals implemented with `libpcap` and raw sockets, to facilitate communication between real-time simulation instances. In the third arrangement, we use the new peering portal design, described in Section 4.2.

The results of this experiment are shown in Figure 8. Here, the aggregate throughput shown is calculated by summing the throughput of all the concurrent flows at each second. To discard the warm-up effect of TCP, we recorded the throughput after after a few seconds when the new TCP flows have been established. The results shows that the performance of our proposed approach is significantly better than the traditional parallel simulation approach, in some cases more than a factor of two. The performance of using the pcap-based emulation portals is comparable to that of the UDP-based peering portals when the traffic is light. When the number of flows increases, the UDP-based peering portals outperform the traditional pcap method.

The capping of the aggregate throughput may be attributed to the TCP congestion control algorithms as well as

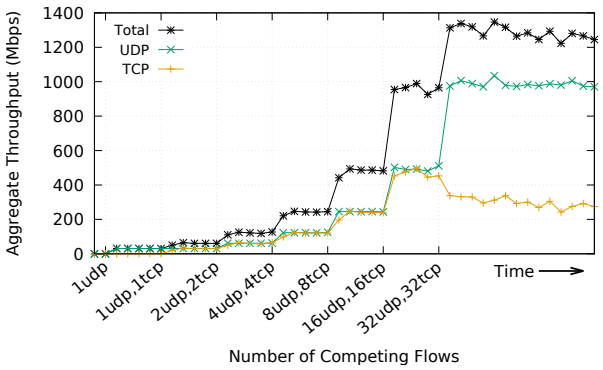

Figure 10: Mixed traffic over peering portals.

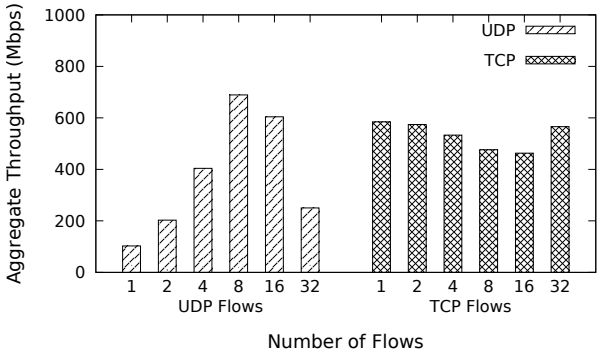

Figure 11: Emulated traffic throughput.

the additional processing cost related to simulating TCP. We conducted another experiment by replacing the TCP flows with UDP constant bit-rate (CBR) flows. Figure 9 compares the performance of emulation portals and peering portals. The peering portals outperform the emulation portals by a margin of almost 2:1 in heavy traffic scenarios.

We conducted the third experiment with mixed TCP and UDP traffic. In this experiment, we start with one CBR UDP flow and then add another TCP flow. After that, we double the number of TCP and UDP flows each time until we reach a total of 64 flows. Figure 9 shows the aggregate throughput of all flows. This experiment demonstrates simulator's capability to support different mix of applications. We notice that the TCP aggregate throughput drops for 64 flows. We believe that this is caused by the competing UDP traffic.

5.3 SDN-Capable Emulation Experiments

To evaluate our proposed parallel real-time simulation infrastructure for supporting SDN experiments, we set up an experiment to recreate the scenario we studied previously with Maxinet (in Section 2). We have 128 hosts evenly partitioned between two physical machines, 64 hosts each. These hosts are implemented as virtual machines, or Linux namespaces, to be more specific. On each machine, the 64 hosts are connected by a network of software routers implemented as Open vSwitch (OVS) instances. These OVS instances are connected as a binary tree; thus, on each machine we have 63 routers connecting the 64 hosts. The root router at each machine is connected to the simulated network through an emulation portal, as depicted in Figure 4. Finally, one of the routers from the simulated network at one real-time simulation instance on one machine connects with the corresponding router at the other real-time simulation instance at the other machine through the peering portal.

During the experiment, we set up 64 pair-wise flows between the hosts on one machine and the corresponding hosts on the other machine. We designate 32 pairs to run TCP flows and 32 pairs to run UDP flows. We use `iperf` to create the flows. For UDP flows, We cap the UDP send rate constant at 100 Mbps. We start the experiment with only one TCP flow and one UDP flow, and then double number of each type of flows every 30 seconds during the experiment until all hosts are running data transfers.

Figure 11 shows the aggregate throughput for UDP and TCP flows separately. Compared to the performance from Maxinet (shown in Figure 3), our parallel real-time simulation approach can achieve an overall aggregate throughput (adding both TCP and UDP) by almost a factor of 6 (in the 16-flow case). We see a significant decline in the aggregate throughput for 32 UDP flows, in which case the applications are generating 3.2 Gbps UDP traffic in total, and since each flow needs to go through 4 hops over a series of OVS routers before it is absorbed into the simulated network, the total packet processing demand would add up to 12.8 Gbps, which exceeds the processing capacity offered by the Linux operating system.

6 CONCLUSIONS

Real-time network simulation allows experimenters to design network experiments that include simulated networks with emulated applications and protocols running in real operating environments. Parallel real-time network simulation, by supporting simulation and emulation on parallel platforms, enables at-scale experimentation with large network models potentially with thousands of emulated applications or more, as they can be run across multiple distributed-memory machines. In this paper, we present an alternative design for parallel real-time simulation to increase the data processing capability and data throughput between the parallel simulation instances.

For parallel real-time simulation, we eliminate parallel synchronization by allowing each simulation instances to advance their simulation clocks by pegging onto the local wall-clock time. We also propose a new type of simulation portal, which is the mechanism used by real-time simulator to communicate with an emulated network and with another parallel simulation instance. We conducted extensive experiments to demonstrate that the proposed technique can indeed improve the overall performance of parallel real-time network simulation.

Pinning real-time simulation to local wall-clock time as opposed to applying global synchronization among parallel

instances may introduce errors from imprecision in the time synchronization across the system. The significance of such errors would depend on the specific applications and we would like to quantify their impact with further experiments. We plan to investigate techniques that can potentially reduce the errors, such as by combining with the shared-memory real-time scheduling techniques [13] and by prioritizing real-time events according to their urgency.

Our parallel real-time simulation technique can be readily applied to studying software-defined networks. An initial feasibility study shows that our approach not only enables a hybrid test environment for SDN studies, but also achieves greater performance when compared with the traditional distributed emulation setup. In the future, we will investigate the computational efficiency of the real-time simulator for handling large-scale networks. We plan to further investigate the batching and data compression techniques, especially the ability to differentiate the processing of different types of application traffic, such as delay sensitive and bandwidth intensive applications. We would also like to investigate techniques that can further improve the the simulation performance and resource utilization, using techniques such as DPDK, zero-copy, kernel packet buffering, etc. Currently, we are only able to conduct feasibility studies for SDN. We plan to investigate efforts in developing large-scale experiments for different SDN solutions to prove the utility of our approach in more practical settings.

ACKNOWLEDGMENT

This research is supported in part by the NSF grant CNS-1563883, the DOD grant W911NF-13-1-0157, the DOE LANL LDRD Program, and a USF/FC2 SEED grant.

REFERENCES

[1] Jeff Ahrenholz, Claudiu Danilov, Thomas R. Henderson, and Jae H. Kim. 2008. CORE: A real-time network emulator. In *MILCOM*. 1–7.

[2] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*. USENIX Association, Berkeley, CA, USA, 19–19. http://dl.acm.org/citation.cfm?id=1855711.1855730

[3] CloudLab. 2017. https://www.cloudlab.us/. (2017).

[4] Miguel A. Erazo, Yue Li, and Jason Liu. 2009. 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 & Communities and Workshops (TRIDENTCOM'09)*. 1–10.

[5] Kevin Fall. 1999. Network emulation in the Vint/NS simulator. In *Proceedings of the 4th IEEE Symposium on Computers and Communications (ISCC'99)*. 244–250.

[6] Richard Fujimoto. 2015. Parallel and Distributed Simulation. In *Proceedings of the 2015 Winter Simulation Conference (WSC '15)*. IEEE Press, Piscataway, NJ, USA, 45–59. http://dl.acm.org/citation.cfm?id=2888619.2888624

[7] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P. Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. 2014. SDX: A Software Defined Internet Exchange. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. ACM, New York, NY, USA, 551–562. https://doi.org/10.1145/2619239.2626300

[8] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. 2012. Reproducible network experiments using container-based emulation. In *Proceedings of the8th CoNEXT*. 253–264.

[9] Brandon Heller. 2013. *Reproducible network research with high-fidelity emulation*. Stanford University, CA, USA.

[10] Eric Jo, Deng Pan, Jason Liu, and Linda Butler. 2014. A Simulation and Emulation Study of SDN-based Multipath Routing for Fat-tree Data Center Networks. In *Proceedings of the 2014 Winter Simulation Conference (WSC '14)*. IEEE Press, Piscataway, NJ, USA, 3072–3083. http://dl.acm.org/citation.cfm?id=2693848.2694235

[11] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM Workshop on Hot Topics in Networks*. 19:1–19:6.

[12] Jason Liu. 2008. A primer for real-time simulation of large-scale networks. In *Proceedings of the 41st Annual Simulation Symposium (ANSS'08)*. 85–94.

[13] Jason Liu. 2013. Real-time scheduling of logical processes for parallel discrete-event simulation. In *Proceedings of the 2013 Winter Simulation Conference (WSC'13)*.

[14] Jason Liu and Yue Li. 2008. On the performance of a hybrid network traffic model. *Simulation Modelling Practice and Theory* 16, 6 (2008), 656–669.

[15] Jason Liu, Yue Li, Nathanael Van Vorst, Scott Mann, and Keith Hellman. 2009. A real-time network simulation infrastructure based on OpenVPN. *Journal of Systems and Software* 82, 3 (March 2009), 473–485.

[16] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (2008), 69–74.

[17] D.M. Nicol, Dong Jin, and Yuhao Zheng. 2011. S3F: the scalable simulation framework revisited. In *WSC*. 3288–3299.

[18] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. 2015. The Design and Implementation of Open vSwitch. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 117–130.

[19] PRIME Research Group. 2013. Parallel Real-time Immersive network Modeling Environment. http://www.primessf.net/. (2013).

[20] Hajime Tazaki, Frédéric Uarbani, Emilio Mancini, Mathieu Lacage, Daniel Camara, Thierry Turletti, and Walid Dabbous. 2013. Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments. In *CoNEXT*. 217–228.

[21] Nathanael Van Vorst, Miguel Erazo, and Jason Liu. 2011. PrimoGENI: Integrating Real-Time Network Simulation and Emulation in GENI. In *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS)*. IEEE Computer Society, Washington, DC, USA, 1–9. https://doi.org/10.1109/PADS.2011.5936747

[22] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M.H. Zahraee, and H. Karl. 2014. MaxiNet: Distributed emulation of software-defined networks. In *Proceedings of the 2014 IFIP Networking Conference*. 1–9.

[23] Jiaqi Yan and Dong Jin. 2015. 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 (SOSR '15)*. ACM, New York, NY, USA, Article 27, 7 pages. https://doi.org/10.1145/2774993.2775012

[24] Junlan Zhou, Zhengrong Ji, Mineo Takai, and Rajive Bagrodia. 2004. MAYA: integrating hybrid network modeling to the physical world. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 14, 2 (April 2004), 149–169.