

An Open and Scalable Emulation Infrastructure for Large-Scale Real-Time Network Simulations

Jason Liu, Scott Mann, Nathanael Van Vorst, and Keith Hellman

Department of Mathematical and Computer Sciences

Colorado School of Mines

Golden, Colorado 80401

Emails: {xliu, scmann, nvanvors, khellman}@mines.edu

Abstract—In this paper we present a software infrastructure that embeds physical hosts in a simulated network aiming to create a large-scale real-time virtual network testbed. Our real-time interactive simulation approach combines the advantages of both simulation and emulation by alleviating the burden of model development that is error-prone and therefore increasing fidelity as real systems are included in the simulation. In our approach, real-world distributed applications and network services can run together with the simulator that operates in real time. Real packets are injected into the simulation system and subject to the simulated network conditions computed as a result of both real and virtual traffic traversing the network and competing for network resources. A prototype of the proposed emulation infrastructure has been implemented based on Virtual Private Network (VPN) customized to function as a gateway that bridges traffic between the physical entities and the simulated network. One distinct advantage of our approach is that it does not require special hardware to set up the system, although a tight coupling between the emulated entities and the simulator will undoubtedly improve both performance and accuracy of the system. Furthermore, our emulation infrastructure is flexible, secure, and scalable—merits inherited directly from the VPN implementation. We conducted a set of preliminary experiments to assess the performance limitations of our emulation infrastructure. We also present an interesting example scenario to show the capability of our approach.

I. INTRODUCTION

Over the years, we have seen a major shift in the global network traffic in response to emerging “killer apps” of the Internet. For example, subject to interpretation, one recent study shows that peer-to-peer (P2P) applications currently contribute over 60% of the total Internet traffic [1]. The landscape of computer networks changes rapidly, which poses a considerable challenge to researchers designing the next-generation high-performance networking and software infrastructures. Traditionally, networking research has relied on a variety of tools, ranging from physical testbeds [2], [3], through analytical models [4], to simulation and emulation [5], [6]. In addition, advanced parallel network simulation tools, such as SSFNet [7], aimed at extending the simulator’s capability to execute extremely large network models has helped improve our understanding of the global-scale network phenomena.

However, as the scale of the network we are able to simulate increases, one cannot underestimate the importance of simulation validation, which seems particularly elusive for large-scale network models. There have been efforts in augmenting

the network simulation with emulation capabilities, which we call real-time interactive network simulation [8]. With the emulation extension, a network simulator operating in real time can run together with real-world distributed applications and network services. These applications and services generate real packets, which are injected into the simulation system. Packet delays and losses are calculated as a function of the simulated network condition. To a certain extent, real-time network simulation not only alleviates the burden of developing separate models for applications in simulation, but also increases the confidence level of network simulation as real systems are included in the network model.

Large-scale real-time network simulation requires simulation be able to characterize the behavior of a network—potentially with millions of network entities and with realistic traffic load. In addition, the simulation must be scalable and able to keep up with the wall-clock time. To this end, parallel discrete-event simulation techniques can be applied to increase the simulation event processing capabilities and to reduce the possibility of missed deadlines [9]. The multi-scale modeling techniques have also been investigated that can selectively include coarse-grain traffic models to effectively reduce the time complexity of a large-scale network simulation [10]–[12].

Scalability also implies high performance of the emulation interface for real applications to interact with the simulator. This is an area where we have not seen significant progress being made. Consider a scenario where collaborative and geographically distributed research teams conduct a remote scientific experiment over the network. The experiment involves physical experimental instruments, supercomputing facilities, and large-scale data storage centers. A high-performance software infrastructure is in place to seamlessly provide coordination between the experimental facilities and the distributed research teams. From the perspective of network design, our goal is to use a large-scale virtual network testbed (presumably run at a supercomputing center) to connect these experimental sites in order to evaluate both software and networking support for this large-scale distributed science application. For networking research, the experimental data—including, for example, intermediate results that need to be transferred and stored, data streams for remote visualization, and control flows for remote experiment steering—provides a realistic traffic load, invaluable for testing new network protocols and

network services. In this scenario, multiple entry points to the network simulator must be provided to facilitate real-time interaction initiated from different geographical locations and with different traffic demand.

This paper presents an emulation infrastructure that facilitates scalable interaction between the real applications and the network simulation. This effort is part of the PRIME (Parallel Real-time Immersive network Modeling Environment) project, designed to provide a self-sustained large-scale virtual network environment for researchers of distributed systems and networks [13]. PRIME is derived from an interactive network simulator called RINSE [14] and aims at addressing several important issues, including:

- Simulation scalability and real-time performance guarantee. We envision that using parallel simulation and multi-scale modeling techniques and combining with novel real-time scheduling algorithms can significantly improve the timeliness of the interactive network simulator.
- Dynamic configuration and continuous network operation. The virtual network testbed is expected to accommodate multiple experiments running either simultaneously or in succession with potentially different network configurations without disruption.
- Scalable interaction and effortless integration with real applications. We need to provide an emulation infrastructure for the network simulator run on supercomputers to dynamically interface with a large number of real applications.

This paper focuses on the last issue. In particular, we propose an emulation infrastructure using a Virtual Private Network (VPN) server farm as the simulation gateway for distributed applications to dynamically connect to the real-time network simulator. Real distributed applications run on computers configured as VPN clients, which automatically forward network packets generated by the applications and targeted the virtual network to the VPN servers. A user-level daemon process run at each simulation gateway manages emulation connections to the simulator and redirects the traffic from the VPN servers to the corresponding simulation processes via TCP connections. At the simulation site, the network packets are translated into simulation events and then injected into the simulator. On the reverse path, packets arriving at an emulated host in the virtual network are translated into real network packets and sent to the simulation gateway via TCP. The user-level daemon process receives the packets and then forwards the packets to the VPN server, which sends the packets to the corresponding real client machine that assumes the identity of the emulated host in simulation.

Using VPN as the simulation gateway between real applications and the simulator brings several benefits:

- 1) The simulation gateway is placed outside the firewall of the supercomputing center where we run the large-scale network simulation. The gateway is necessary because direct emulation connections to the simulator are typically blocked by the firewall. The gateway provides

a natural rendezvous point between the real distributed applications and the simulator.

- 2) On a client machine, each VPN connection is implemented through a network tunnel device assigned with a proper IP address. We use an automatically generated VPN configuration file to customize the IP routing table to allow network traffic targeting the virtual IP address space to be forwarded via the VPN connection. Since in most cases the tunnel device is treated as a regular network interface, this process is therefore transparent to the application we intend to emulate.
- 3) The VPN server is capable of handling a large number of VPN connections. Allowing dynamic behavior is important because we expect the virtual network testbed to run for a long period of time. Individual emulation connections may comparatively have a shorter life span.
- 4) Several VPN servers can be used to balance the traffic going in and coming out of the real-time simulator. A client machine emulating a real application can choose to connect to a VPN server according to its geographical location and/or the quality-of-service demand. One can also adopt a load balancing strategy to select a simulation gateway in a VPN server farm.
- 5) VPN provides support for enterprise-scale configurations, such as fine-grain access controls, authentication, and encryption. It also provides fault-tolerance measures, such as fail-over. Data compression is also used for packet transmission through the VPN connection for better performance. We inherit all these features automatically.
- 6) VPN packages are in general highly portable. For example, OpenVPN, the implementation we use for the simulation gateway, is open source and freely available on Windows, MacOS X, and most Unix platforms [15].

The remainder of this paper is organized as follows. In section II, we describe previous work related to our approach. Section III presents our emulation infrastructure in detail. We implemented a prototype of this system to study the feasibility of such an approach. To demonstrate the performance of this emulation infrastructure and assess its limit, we conducted preliminary experiments on a relatively restricted physical testbed. We report the results in section IV. We also used this emulation infrastructure to interact with existing network applications, as well as a real commercial network device designed for content filtering particularly in cyber-security applications. Section V presents an example. We conclude this paper with a discussion of future work in section VI.

II. RELATED WORK

Advanced emulation techniques can now provide full-scale emulation capabilities for large-scale networks [6], [16], [17]. For example, ModelNet [6] multiplexes real network applications at the peripheral of the system called “edge nodes”. Multiple applications run unmodified on each edge node and, by setting up explicit routes, send traffic through the emulation core that runs on parallel computers. Network operations are

emulated using a time-driven approach to calculate packet delays and losses. At its core, ModelNet uses an IP firewall (`ipfw`) rule to intercept incoming packets and a kernel module is then used to simulate packet forwarding in the network. The ModelNet approach is appropriate for a closed emulation system, where edge nodes are configured statically as an integral part of the system. A careful setup of the physical testbed with a tight coupling between the edge nodes and the core can achieve a very high throughput allowing the emulation to scale up to tens of thousands of network entities. Our emulation infrastructure instead is open and flexible, allowing dynamic connections to the virtual network testbed—the association between real applications and the simulated network is *ad hoc* and potentially transient. More important, aside from conducting traffic generated by the emulated applications, our network simulator can simulate additional traffic in the background providing a more realistic and yet controllable network condition.

Bradford et al. gave a survey of techniques for importing and exporting real network packets, such as using raw sockets and the `pcap` library, that allow real-time interactions with the network simulator [18]. In MaSSF [19], applications can also run unmodified together with the simulator. This is accomplished by intercepting communication-related and time-sensitive systems calls, either through the use of linker wrapper functions or with the replacement of dynamic linking libraries. To enable interaction with the emulation system, these techniques invariably require the applications to receive some special treatment, either at compile/link time, which is problematic if one does not have access to the source code, or at run time, which tends to be difficult to implement and maintain. Furthermore, these techniques do not address issues, such as scalability and load balancing, which we regard as important for a large-scale emulation system.

Our approach extends the previous effort in the RINSE simulator, which also allows real-time interactions with real applications presumably running on geographically distributed client machines [14]. RINSE prescribes a blueprint for a large-scale emulation system. In particular, the design of RINSE is the first we know to designate a data server to connect the client applications with the real-time simulator. By using a database, the server can perform authentication and event bookkeeping for client applications accessing the simulated network. The emulation infrastructure we describe in this paper presents a viable approach to large-scale network emulation, which not only supports an efficient and flexible pathway to the real-time simulator, but also provides a scalable solution for managing client applications.

III. THE EMULATION INFRASTRUCTURE

As illustrated in Fig. 1, the emulation system consists of three major components:

- 1) The I/O threads are collocated with the real-time network simulator run on parallel computers (most likely in a supercomputing center behind a firewall). The threads are connected to the simulation gateways, configured

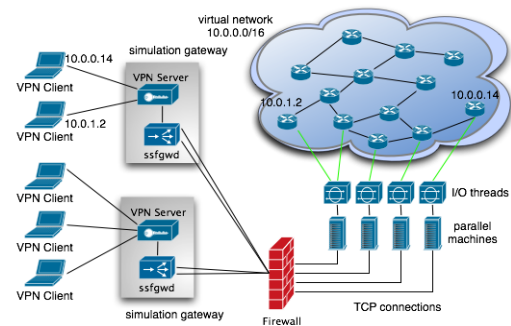


Fig. 1. The Emulation Infrastructure

to tunnel network packets to and from the applications running on client machines.

- 2) Each simulation gateway runs a VPN server and a daemon process called `ssfgwd`, which is responsible for shunting IP packets between the virtual network and the client applications.
- 3) The client machines each run a VPN client configured to connect to a designated simulation gateway (which can be determined dynamically for load balancing among other criteria). Presumably there is a one-to-one correspondence between a client machine and an emulated host inside the simulation. VPN sets up the IP address for the client machine using the same IP address of the emulated host. Furthermore, the IP routing table of the client machine is also set up automatically for the client applications to forward traffic to the virtual network through the VPN connection.

In the remainder of this section, we describe the details of these three components, as well as a mechanism to emulate routers with multiple network interfaces.

A. The Real-Time Network Simulator and the I/O Threads

PRIME extends the SSFNet network simulator for real-time large-scale network simulations. SSFNet was developed based on a simulation kernel that implements the Scalable Simulation Framework (SSF) for high-performance parallel and distributed discrete-event simulation [20]. The simulator has been augmented to allow real-time interactions with external network applications.

A virtual network is partitioned among parallel processors to be simulated in parallel. Each processor participating in the parallel network simulation spawns two threads in addition to the simulation process assigned to this processor for event processing. An I/O agent in the simulation process is responsible for both exporting packets from and importing packets to emulated hosts in simulation. A writer thread takes the packets exported from the simulation process and forwards them to a simulation gateway, which is then responsible for delivering the packets to the corresponding client machine. A reader thread accepts packets from the client applications through a simulation gateway and inserts the packets into the simulator.

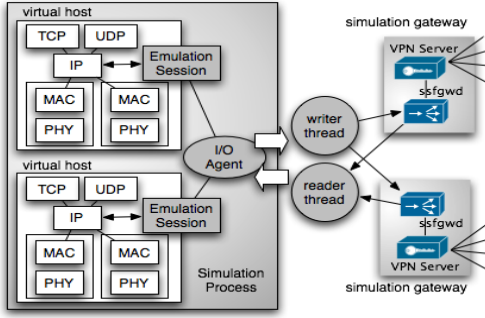


Fig. 2. Connecting the Simulator and the Simulation Gateway

The I/O agent and both I/O threads use established SSF functions designed specifically to support emulation, including both exporting simulation events and importing real network packets (see [14] for more details). Each emulated host in the simulation contains an emulation protocol session that intercepts packets at the IP layer. Upon receiving a packet event deemed to be sent out to the client application, the emulation session forwards the event to the designated I/O agent in the same simulation process (on the same processor), which exports the event. Outside the simulator, the writer thread sends the IP packet (translated from the simulation event) to the simulation gateway via TCP. On the reverse path, when the reader thread receives an IP packet, it translates the packet to a simulation event and presents it to the I/O agent inside simulation. The I/O agent then forwards it to the emulation protocol session on the corresponding virtual host, which pushes the packet down the simulated protocol stack. This procedure is illustrated in Fig. 2.

B. The Simulation Gateway

The simulation gateway is a critical component of the emulation infrastructure, residing between the network simulator and the client applications run on distributed machines. We set up a VPN server at each simulation gateway managing incoming connections from the client machines. There can be multiple simulation gateways for balancing the traffic load or for differentiated services. At each gateway, traffic to and from the client applications are handled by VPN. We defer the details of the VPN setup to the next section.

In addition to running the VPN server that manages the VPN clients, we create another process at the simulation gateway to handle traffic to and from the network simulator. The `ssfgwd` daemon is a process responsible for shunting IP packets between the simulator's I/O threads and the VPN server. It maintains a separate TCP connection with each of the I/O threads on the parallel machines running the simulation. It is important that the TCP connection must be initiated from the simulator, since the simulator is expected to run at the back-end of a supercomputer, which is situated typically behind a firewall and cannot be determined *a priori*. It is quite possible that incoming TCP connections to the supercomputer are blocked by the firewall rules.

The type of the TCP connections to a simulation gateway is distinguished by the data subsequently sent from the I/O threads via the TCP connections. A one-byte identifier is used to label a connection to either a reader thread or a writer thread. In addition, the IP addresses of all virtual hosts to be emulated by the corresponding simulation process are subsequently sent by the reader thread. The `ssfgwd` process records this list of IP addresses and creates a mapping from the IP addresses to the corresponding TCP connection to the reader thread that sends these addresses. Later, `ssfgwd` uses the mapping to forward traffic from the client machines to the corresponding simulation processes.

In order to support emulation of hosts with multiple network interfaces (for reasons which we discuss in section III-D), we modified the VPN server to spawn the `ssfgwd` process directly and communicate with it using the standard Unix pipes. An IP packet received by the VPN server from one of its clients is preceded with the client's IP address before it is sent to `ssfgwd` via the pipe. The IP address is used by `ssfgwd` to deliver the packet to the corresponding simulation process. Recall that a mapping from the IP address to the simulation process that contains the client's virtual host is established immediately after the TCP connection is made by the reader thread. Once the TCP connection is located, the packet is sent via the TCP connection to the designated reader thread at the simulator. The packet is subsequently inserted into the protocol stack of the emulated host that carries the same IP address of the client machine that initiates this packet. In such a way, the packet appears as if it were generated directly by the virtual host. On the other hand, an IP packet emanating from a virtual host is sent to the simulation gateway through the TCP connection established by the writer thread. Again, the packet is preceded with an IP address identifying the virtual host (more precisely, a network interface on the virtual host) that exports the packet. The `ssfgwd` process sends both the IP address and the packet via the pipe to the VPN server, which forwards the packet to the corresponding VPN client according to the IP address. In such a way, the packet arrives at the client machine as if it receives the packet directly from a physical network.

We can opt to use multiple simulation gateways to alleviate the traffic load placed otherwise on a single gateway forming a bottleneck. Such load balancing decisions can be made either statically or dynamically. All emulated hosts in the virtual network can be partitioned and assigned to different simulation gateways at the configuration time. The entire emulation traffic is therefore divided among the simulation gateways and a better throughput is anticipated. Alternatively, the clients may dynamically choose among a set of simulation gateways at the time of its connection. For example, the IP Virtual Server (IPVS) can be used to implement a rather sophisticated load balancing scheme at a front-end server, which subsequently redirect services to a cluster of servers at the back-end [21]. In other situations, the clients may choose to connect to a gateway that can satisfy a particular quality-of-service demand. This allows us to differentiate client traffic to and from the real-time

simulator. The quality-of-service demand could be as simple as finding a simulation gateway that is geographically close to the client, or more complicated as to find a simulation gateway that can sustain traffic with certain throughput and latency requirements. In our implementation, we adopted the OpenVPN’s simple load-balancing scheme having the clients to choose randomly from a set of simulation gateways at the time of connection. In any case, the I/O threads at the simulator must make separate TCP connections to all simulation gateways. We implemented mechanisms to allow a client each time to connect to a different gateway so that traffic from the simulator can be routed to the client machine through the gateway with which the client is currently engaged.

C. The VPN Connections

OpenVPN [15] was chosen to allow applications to dynamically connect to the simulation gateway and to emulate network interfaces on the client machines. We chose OpenVPN in our implementation, since it is publicly available and runs on most operating systems. In our scheme, one runs a modified OpenVPN server on each simulation gateway using an OpenVPN server configuration file automatically generated from the virtual network specification. Each OpenVPN client connects to a chosen simulation gateway using another client configuration file, also generated automatically beforehand. Each client will have an IP address assigned to the virtual private network interface that matches the IP address of the emulated network interface in simulation so that the client machine can assume the proper identity as applications running on this machine can transparently forward traffic to the simulated network.

As in SSFNet, the virtual network is specified by a simple configuration script in the Domain Modeling Language (DML). A DML network description includes the topology of the network (i.e., sub-networks, hosts, routers, and links connecting network interfaces in the hosts and routers), the network protocols running at each host or router, and the traffic traversing the network (see [20] for details). All virtual network interfaces defined therein are then automatically assigned with an IP address using a utility program called `dmlenv`, which creates another DML file that contains auxiliary information in addition to the automatically assigned IP addresses, including the mapping of communication channels among simulation entities and the alignment of hosts and routers to form simulation processes—all necessary for setting up the network simulation. The auxiliary information is also given to `dmlpart`, another utility program that partitions the virtual network and generates yet another DML file describing the assignment of the simulation processes to parallel processors. All these DML files are needed when we start the network simulator.

We wrote another utility program called `vpnscrip`t to automatically generate the configuration files needed by the OpenVPN servers and clients. We use a server configuration file to set up each simulation gateway and a client configuration file for each emulated network interface. The

`vpnscrip`t first creates a certificate authority used by the OpenVPN servers to validate the encrypted incoming client connections. The script then generates a pair of public and private keys for each OpenVPN server or client. Finally, `vpnscrip`t creates a compressed archive file that includes all information to start an OpenVPN server or client. For each simulation gateway, the archive includes the private key of the OpenVPN server and the public keys of all OpenVPN clients managed by this simulation gateway (each corresponding to an emulated network interface). Also included in the archive is a mapping from public keys to IP addresses. The mapping is used by the OpenVPN server to statically assign IP addresses to incoming client connections, which are distinguished by the clients’ public keys. For each OpenVPN client, the archive includes the private key of the client and the public keys of all designated simulation gateways. For convenience, a script is included in these archives to help start the OpenVPN servers and clients automatically.

We modified the OpenVPN implementation to allow the simulation gateway to correctly forward IP packets to and from a client machine emulating a router (i.e., with multiple network interfaces). We postpone the discussion of the emulation infrastructure’s IP forwarding capability to the next section. We also added a plug-in to the OpenVPN server to notify the network simulator of the connection status of the client machines. When a client establishes a VPN connection, the plug-in code will be run to send a notification packet to the emulation protocol session on the corresponding virtual host. Upon receiving this notification, the virtual host is then allowed to export packets to the client machine. Similarly, when a client machine disconnects from the server, a notification is sent from the plug-in to the simulator so that simulated packets to the virtual host will be dropped at the simulator rather than carried all the way to the simulation gateway and then dropped.

D. IP Forwarding

Each VPN connection at the client machine corresponds to an emulated network interface inside the simulation. For multi-homed hosts, multiple VPN connections are needed. It would not be necessary to modify the VPN server if these emulated hosts were simply end hosts. In this case, `ssfgwd` could run independently from the VPN server and they could communicate using standard IP tunnel devices. For example, immediately upon startup, `ssfgwd` could open a tunnel device, activate and configure the device by assigning a special IP address to it, and then modify the kernel routing table to connect the tunnel device with the one created by the VPN server. The kernel routing table could be set up so that packets arriving from the VPN server’s tunnel device be forwarded by the kernel to `ssfgwd`’s tunnel device and similarly packets written by `ssfgwd` to its tunnel device be forwarded to the VPN server, which are then delivered to the corresponding clients. Packets arrived at the simulation gateway from a client machine could be sent to the simulation process and the corresponding virtual host according their source IP addresses. Similarly, packets arrived at the simulation gateway from

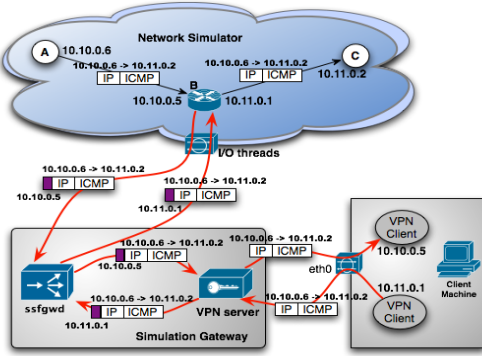


Fig. 3. Emulation of a Router with Multiple Network Interfaces

the simulator could be distinguished by their destination IP addresses and sent to the corresponding client machines via the corresponding VPN connections.

This scheme, however, does not work if one wants to emulate a router with multiple network interfaces (e.g., to test a real routing protocol implementation). Consider an example shown in Fig. 3, where we emulate the router B in a virtual network of three nodes. Since router B has two network interfaces with IP addresses 10.10.0.5 and 10.11.0.1, respectively, we run two separate VPN clients on the client machine. Suppose that node A pings node C . An IP packet with a source address of 10.10.0.6, a destination address of 10.11.0.2, and a payload of an ICMP ECHO-REQUEST message reaches the network interface 10.10.0.5 and is then exported to the simulation gateway as expected. Since the packet carries a source address of 10.0.0.2, which is not emulated, the `ssfgwd` process will have trouble forwarding the packet to the client machine.

To overcome this problem, each packet sent between the network simulator and the simulation gateway is preceded with an IP address indicating the network interface with which the packet is associated. For example, when exporting the ICMP packet arrived at the network interface 10.10.0.5, the writer thread sends the address 10.10.0.5 ahead of the packet to the simulation gateway. The VPN server is modified to use this leading address to differentiate traffic to the client machines,¹ in this case, forwarding the ICMP packet to the client machine labeled as 10.10.0.5 through the corresponding VPN connection. The client machine therefore receives the ICMP packet from the logical network interface 10.10.0.5 exactly as it happens in the simulated network. If the routing table of the client machine is set up correctly, the packet will be sent out immediately by the kernel from the logical network interface 10.11.0.1. The VPN server, upon receiving this packet, will attach a prefix address 10.11.0.1 to the packet before sending it onward to the `ssfgwd` process and subsequently to the simulator. The reader thread at the simulator uses this leading address to dispatch the packet to the corresponding virtual

¹Note that the changes only apply to the VPN server. In our implementation, the modified OpenVPN program must be used as part of the simulation gateway. The VPN clients, however, still use a standard VPN distribution.

host. The ICMP packet is then sent down the protocol stack of router B continuing its journey on the virtual network.

IV. EXPERIMENTS

We ran a set of preliminary tests to gauge the performance limitations of our emulation infrastructure. In particular, we assess the impact on the accuracy of the emulation as it inevitably introduces overhead when network traffic is sent between applications running on the client machines and the real-time simulator through the simulation gateway. The overhead, manifested as packet delays and losses that are not part of the simulated network, is a major contributor to the emulation errors. In our particular experiment setup, the machines running the VPN clients and the simulation gateway are both AMD Athlon64 machines (with a 2.2 GHz CPU and 2 GB of memory) connected via a gigabit switch. We chose two sites to run the network simulator. One is a 23-node Linux cluster located on campus (each node with a Pentium IV 3.2 GHz CPU and 512 KB of memory). The campus network is a 100 Mb/s Ethernet. The other site we chose to run the simulation is an IBM terascale supercomputer named DataStar (consisting of 272 8-way p655+ and 7 32-way p690 compute nodes) located at the San Diego Supercomputer Center [22].

The simulated network is shaped like a dumbbell as shown in Fig. 4, which has been commonly used to evaluate TCP congestion control algorithms. There are N nodes aligned on either side of the dumbbell connected by two routers. A server node on one side of the dumbbell is sending traffic via TCP to the corresponding client node on the other side of the dumbbell. The delay of each branch link connecting a router with one of its adjacent client or server nodes is set to be 100 milliseconds and its bandwidth is 100 Kb/s. The link connecting the two routers in the middle of the network has a delay of 300 milliseconds and a bandwidth of $100 \times N$ Kb/s. In our experiments, we emulated router R_1 , which is located on the server side of the dumbbell network. On the client machine, we simultaneously run several VPN clients, each corresponding to a network interface within router R_1 . We also enabled IP forwarding on the client machine and set the kernel routing table accordingly to forward packets to and from the VPN tunnel devices. This dumbbell network was designed to expose the overhead of the emulation infrastructure. By changing the number of TCP client/server pairs we could vary the amount of network traffic traversing the emulation infrastructure. Although our network simulator supports parallel execution, the simulation here was trivial enough to be run sequentially.

Before the TCP transfers began, we first ran `ping` on both the machine running the simulator and the machine running the VPN clients targeting the simulation gateway to measure the round-trip times of the segments between the simulator and the simulation gateway (represented as x in Table I) and between the simulation gateway and the client machine (represented as y). In the mean time, the client machine, on behalf of router R_1 , was pinging one of the simulated server nodes (say, S_1 , with the result being represented as z). Let z' be the simulated

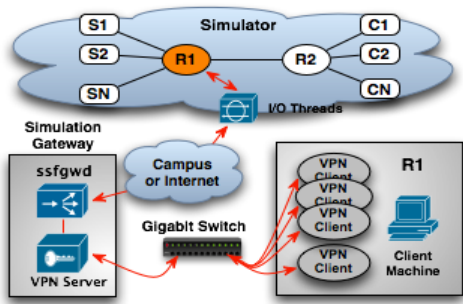


Fig. 4. Emulation of a Dumbbell-Shaped Network

TABLE I
ROUND-TRIP TIMES MEASURED IN MILLISECONDS

	Local Linux Cluster	SDSC Supercomputer
x	0.210 ± 0.064	46.674 ± 3.383
y	0.042 ± 0.007	0.045 ± 0.007
z	200.794 ± 0.889	247.197 ± 2.827
ϵ	0.39%	23.59%
δ	0.529	0.464

round-trip time between router R_1 and S_1 when R_1 was not emulated (200.013 milliseconds in this case). We calculate the emulation error to be $\epsilon = |z - z'|/z'$. Also, the difference in the round-trip times $\delta = z - z' - x - y$ reflects the overhead in addition to the network latencies of the two segments—including, for example, the time it takes to import and export simulation events and to transport packets between `ssfgwd` and the VPN server. The baseline measurement reported in Table I was collected during a relative network quiescence. The results shown are averages of 10 separate runs. In both case, the delay overhead from the emulation infrastructure (δ) accounts for only about 1/2 milliseconds.

Next we allowed all server nodes to each send a file of 300 KB simultaneously over TCP to their client counterparts. We varied the number of client/server pairs, N , during the experiment. Figures 5 and 6 show the amount of data transferred over all TCP connections as we ran the network simulator on the local Linux cluster and at the SDSC supercomputing center, respectively. Scaled up to 64 TCP connections, the emulation on the local cluster produced results closely matching the expected behavior. The telltale staircase shape of the curve on the left of the figure is typical of a TCP transfer. The maximum height of staircase shows that there were at most 64 segments in flight at the peak transfer, which can be translated to an aggregate throughput of 32 Mb/s for all 64 TCP connections. As we increased N to 128, the performance diverged. Here we show the amount of data transferred during the first 40 seconds for each of the 10 runs on the right of Fig. 5. As the traffic demand went beyond the empirical capacity of the 100 Mb/s campus network, significant and unpredictable packet losses occurred causing the TCP congestion control to reduce the

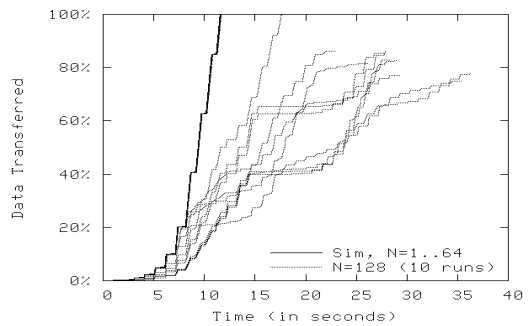


Fig. 5. Emulation of TCP Data Transfers on Local Linux Cluster

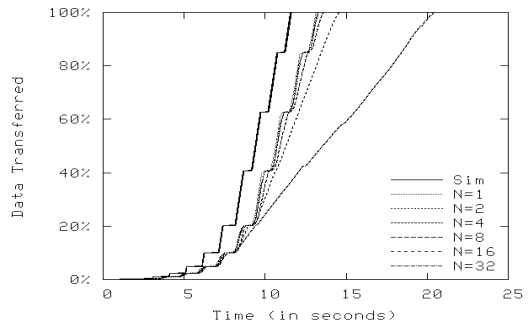


Fig. 6. Emulation of TCP Data Transfers on SDSC Supercomputer

send rates accordingly—a phenomenon that would not have occurred in the simulated network. The emulation result in these cases deviates from the expected network behavior.

Fig. 6 shows the result of the network simulator running on the SDSC supercomputer. Even with a single TCP connection, the overall throughput decreased markedly to 88% of the expected value, mainly due to the significant round-trip time between the server node and the client node. More TCP connections further exacerbate this problem as additional traffic lengthened the delays experienced by the packets crossing the emulation infrastructure. Different from the previous scenario, the latency—rather than packet loss—played a major role in admitting the emulation errors. As shown in Fig. 7, the throughput reduced to 57% for 32 TCP connections, placing the outcome of the emulation poignantly different from the expected behavior.

We should emphasize here that the exact results from these experiments are unimportant since they merely reflect a particular setup of our emulation system. The experiments, however, show that the losses and delays experienced by the packets as they travel through the simulation gateway and the client machines can have a profound impact on the emulation accuracy. On the one hand, to avoid the damaging effect from network latency depicted in Fig. 6, the simulator should be placed close to both the simulation gateway and the client machines. This is especially important if we are to support applications that are particularly sensitive to network latencies (such as ping). On the other hand, with a tight coupling provided between the simulator, the gateway, and the

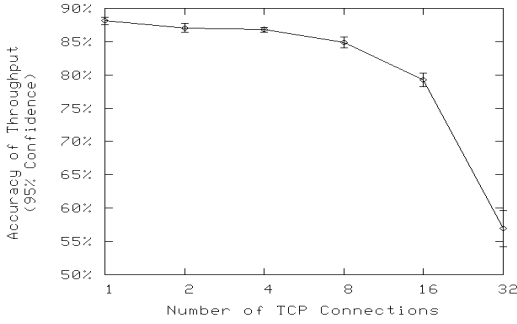


Fig. 7. Accuracy of TCP Throughput on SDSC Supercomputer

client machines, the latency overhead is insignificant. More important, our experiments confirm that the throughput of the emulation infrastructure can scale all the way up the network’s physical limit.

It is for certain that sufficient bandwidth must be provided to allow the system to sustain further emulation traffic in a more demanding scenario. However, in cases where no reference to the wall-clock time is required (as in the case study of the following section), we can allow the real-time simulator to slow down. That is, we fix the virtual time advancement to only a constant fraction of the real time, that is, the simulation time advances by $f \leq 1$ seconds for each wall-clock second. In relative terms, this essentially increases the packet processing capability of the simulation gateways and the client machines, as well as the capacity of the physical network connections—by a factor of $1/f$.

V. A CASE STUDY

In this section we show a study that uses the emulation infrastructure to test a network device named Content Aware Policy Engine (CAPE) currently being developed at Northrop Grumman. CAPE allows one to use a simple language to examine, search, log, and modify the content of the packets that pass through the device. CAPE has both hardware and software implementations. For this case study we used a software version that runs on a Linux box. The program, written in python, uses the Berkley packet filter to snatch packets from the logical interfaces and then applies rules to the packets, possibly modifying them as specified in the simple language before forwarding the resulting packets onward. CAPE uses ipchains to drop the original packets so that the host OS is prevented from processing them unintentionally.

In this study we used CAPE for content routing and distribution. We set up a simulation network that consists of two campus subnetworks with a total of 1008 hosts and routers. This network is a scaled-down version of the baseline network model from the DARPA NMS program, which is used commonly for large-scale simulation studies. Each subnetwork (shown in Fig. 8) has a server cluster (in net 1) that acts as the traffic source. Also, in net 2 and net 3, there are a total of 12 local area network clouds, each with 42 client hosts acting as the traffic sink. To populate the network with sufficient

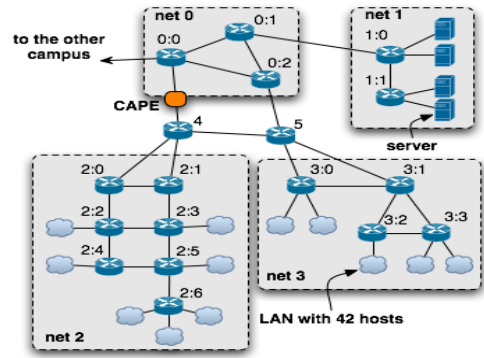


Fig. 8. The DARPA NMS Campus Network Instrumented with CAPE

background traffic, at each client network cloud, we designated 320 fluid TCP flows, 2 packet TCP flows, and 2 packet UDP flows, each downloading data from a randomly selected server from both campuses (see [12] for more detail on the fluid TCP model). The application was streaming video, simulated as a sequence of 1 KB UDP datagrams sent at a constant rate. We randomly chose N clients in net 2 of one campus network having them each request a 100 KB/s video stream from a server randomly chosen from the other campus network. We placed the CAPE device between the routers labeled 0:0 and 4 so it could intercept the streaming traffic. We measured the overall average packet loss rate of all clients as a indication of the receiving video quality.

We ran three tests. As the baseline, we took out the CAPE device and simply ran the simulation. For comparison, in the second test we emulated the CAPE device but programmed it to forward all the traffic without inspecting the content. In the third test we configured the CAPE device to allow only one outstanding request from a client to reach the server and the rest were cached inside the CAPE device. When the server streamed the video back to the client, CAPE replicated the video stream to all other clients who also requested the video download. In this way, the network path between the CAPE device and the server was only burdened with one video stream and better performance was expected.

We ran the experiments on the same AMD machines with a gigabit connection as in the previous section. We varied the number of clients N from 5 to 160, resulting an aggregate traffic of 4 to 128 Mb/s. We found the python-based software CAPE device was unable to process packets at a rate higher than 4K packets/s without significantly dropping them. Therefore, for experiments with more than 40 clients, the speed of the real-time simulator was throttled down to accommodate the slow processing of the CAPE device. Figure 9 shows that, as expected, the packet loss rate (averaged over 10 runs) increases with the increase of clients when content distribution is disabled. The results from simulation and emulation are statistically indistinguishable. When replication is activated in CAPE, however, the loss rate remains stable, implying that the packet losses occur primarily at the network segment between

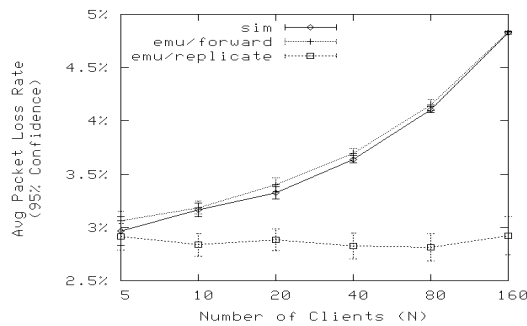


Fig. 9. Using CAPE Content Distribution to Reduce Packet Loss

the CAPE device and the streaming video server.

VI. CONCLUSIONS AND FUTURE WORK

An important aspect of our emulation infrastructure is the use of an existing VPN framework, thus allowing us to inherit its important properties to derive a flexible, portable, scalable, and secure implementation of a simulation gateway between the network applications and the real-time simulator. Alternatively, one could modify kernel modules to arrive at the same functionality and possibly with marginally better performance, albeit at the cost of less portability and maintainability.

Our implementation allows client machines to choose randomly among multiple simulation gateways (i.e., a VPN server farm) to connect to the simulator. Such choices can be altered during the simulation. That is, a client machine can sever a connection at run time and then reconnect to the simulator via another simulation gateway. Although this arrangement is expected to improve the scalability of the emulation infrastructure, we did not conduct any extensive test due to the limitations of our current physical testbed. We plan to conduct further scalability tests in the near future. We also plan to investigate other criteria that can be used by client machines to select among multiple simulation gateways. For example, client connections can be differentiated by the bandwidth and latency requirements: a client running applications with a more stringent quality-of-service requirement should be assigned to a simulation gateway with higher bandwidth and lower latency. Furthermore, we plan to investigate dynamic traffic load balancing schemes and fault tolerance measures.

Our emulation infrastructure will be used as the basic building block of an emulation testbed where physical networks can fully interact with virtual networks in a truly immersive network environment for prototyping, testing, and evaluating network applications, protocols, and services. Our work reported in this paper is only the beginning.

ACKNOWLEDGMENTS

This research is supported in part by a National Science Foundation Career grant CNS-0546712. The authors would like to thank the San Diego Supercomputer Center (SDSC) for allowing access to the computing resources and Northrop Grumman Corporation for the CAPE device prototype. And

thanks to Phil Romig and Mike Colagrosso, who helped set up the physical testbed.

REFERENCES

- [1] CacheLogic, Inc, "Peer-to-peer in 2005." [Online]. Available: <http://www.cachelogic.com/research/index.php>
- [2] P. Barford and L. Landweber, "Bench-style network research in an Internet instance laboratory," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 21–26, 2003.
- [3] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," in *Proceedings of the 1st Workshop on Hot Topics in Networking (HotNets-I)*, October 2002.
- [4] Y. Liu, F. L. Presti, V. Misra, D. F. Towsley, and Y. Gu, "Scalable fluid models and simulations for large-scale IP networks," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 14, no. 3, pp. 305–324, July 2004.
- [5] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in network simulation," *IEEE Computer*, vol. 33, no. 5, pp. 59–67, May 2000.
- [6] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker, "Scalability and accuracy in a large scale network emulator," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002.
- [7] J. H. Cowie, D. M. Nicol, and A. T. Ogielski, "Modeling the global Internet," *Computing in Science and Engineering*, vol. 1, no. 1, pp. 42–50, January 1999.
- [8] D. M. Nicol, M. Liljenstam, and J. Liu, "Advanced concepts in large-scale network simulation," in *Proceedings of the 2005 Winter Simulation Conference (WSC'05)*, December 2005.
- [9] R. Simmonds and B. W. Unger, "Towards scalable network emulation," *Computer Communications*, vol. 26, no. 3, pp. 264–277, February 2003.
- [10] D. M. Nicol and G. Yan, "Discrete event fluid modeling of background TCP traffic," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 14, no. 3, pp. 211–250, July 2004.
- [11] J. Zhou, Z. Ji, M. Takai, and R. Bagrodia, "MAYA: integrating hybrid network modeling to the physical world," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 14, no. 2, pp. 149–169, April 2004.
- [12] J. Liu, "Packet-level integration of fluid TCP models in real-time network simulation," *Proceedings of the 2006 Winter Simulation Conference (WSC'06)*, December 2006, to appear.
- [13] —, "The PRIME Research." [Online]. Available: <http://prime.mines.edu/>
- [14] M. Liljenstam, J. Liu, D. M. Nicol, Y. Yuan, G. Yan, and C. Grier, "RINSE: the real-time interactive network simulation environment for network security exercises," in *Proceedings of the 19th Workshop on Parallel and Distributed Simulation (PADS'05)*, June 2005, pp. 119–128.
- [15] J. Yonan, "OpenVPN – an open source SSL VPN solution." [Online]. Available: <http://www.openvpn.net/>
- [16] P. Zheng and L. M. Ni, "EMPOWER: a network emulator for wireline and wireless networks," in *Proceedings of the IEEE INFOCOM 2003*, vol. 3, March/April 2003, pp. 1933–1942.
- [17] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002, pp. 255–270.
- [18] R. Bradford, R. Simmonds, and B. Unger, "Packet reading for network emulation," in *Proceedings of the 9th International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*, August 2001, pp. 150–157.
- [19] X. Liu, H. Xia, and A. A. Chien, "Network emulation tools for modeling grid behavior," in *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03)*, May 2003.
- [20] SSF Research Network, "Modeling the global Internet." [Online]. Available: <http://www.ssfnet.org/>
- [21] IP Virtual Server (IPVS). [Online]. Available: <http://kb.linuxvirtualserver.org/wiki/IPVS>
- [22] San Diego Supercomputer Center at UCSD, "SDSC DataStar user guide." [Online]. Available: http://www.sdsc.edu/user_services/datastar/