SAND2021-8495C

Case Studies in Experiment Design on a minimega-based Network Emulation Testbed

Brian Kocoloski, USC ISI
Alefiya Hussain, USC ISI
Matthew Troglia, Sandia National Laboratories
Calvin Ardi, USC ISI
Steven Cheng, Sandia National Laboratories
Dave Deangelis, USC ISI
Christopher Symonds, Sandia National Laboratories
Michael Collins, USC ISI
Ryan Goodfellow, USC ISI
Stephen Schwab, USC ISI

ABSTRACT

Testbeds are useful platforms for deploying realistic cybersecurity experiments in controlled environments. Our team recently used minimega, a network emulation system using node and network virtualization, to support evaluation of a set of networked and distributed systems for topology discovery, traffic classification and engineering. This paper presents our approach to enable rapid and rigorous evaluations for these systems and captures our experience, lessons learned and takeaways from the experimentation on our minimega based network emulation testbed. We present the methodology we adopted to encode network and traffic definitions into an experiment description model, and how our tools compile this model onto the underlying minimega API. We then present three cases studies which highlight our solution's ability to support network topological diversity, diverse traffic mixes, and networks with specialized layer-2 connectivity requirements. We conclude with takeaways and lessons learned during our evaluation process.

ACM Reference Format:
Brian Kocoloski, Alefiya Hussain, Matthew Troglia, Calvin Ardi, Steven Cheng, Dave Deangelis, Christopher Symonds, Michael Collins, Ryan Goodfellow, and Stephen Schwab. 2021. Case Studies in Experiment Design on a minimega-based Network Emulation Testbed. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Network emulation testbeds [7, 15, 26, 32, 34] have been widely used for the last two decades to evaluate systems as they allow users to deploy experiments in realistic and controlled environments. Testbeds allow users to deploy networks and nodes capable of running real unmodified software, including operating systems, routing/switching libraries, VPN and firewall programs, etc., as well as end-host traffic from real applications. The ability to run unmodified code provides fidelity and allows researchers to generalize systematic results to the real world.

In recent years, node and network virtualization techniques have further improved the utility of testbed based systems, as they allow users to deploy arbitrary layer-2 network topologies and run experiments at large scales that are not as constrained by the availability of physical nodes and networks. The minimega [21] system from Sandia National Laboratories exemplifies the use of virtualization techniques, allowing it to emulate large and complex networks on physical systems ranging from a single laptop to large-scale clusters and supercomputers. Recent work has demonstrated that while minimega's use of virtualization does lead to some variance in low level network behavior (e.g., packet jitter), most core application and OS behaviors are generally similar in virtual and physical environments [8].

We needed to evaluate novel network analysis and traffic engineering research projects for the DARPA SearchLight program [18]. Due to its flexibility and high fidelity, we selected minimega deployed on the DETER testbed [34] to support our evaluation. Though minimega proved to be a very useful tool, we found the process of rapidly defining

and automating experiments to be challenging on our minimega enabled testbed. Generally, we found that our challenges stemmed from minimega's approach to be both highly configurable and able to emulate many different network elements with high fidelity. In order to meet these goals, minimega's API is low level; it gives users control to emulate specific processor models/ISAs, network interface cards (NICs), and many other device and architectural features that could be needed to achieve various timing and behavioral characteristics. For example, some DPDK [24] applications may desire virtio [28] or vfio [33] rather than minimega's default e1000 NICs, or advanced device features such as multiqueue RX/TX ports. minimega also gives users extensive control over the experiment runtime environment with its command-and-control interface, `miniccc` [19], which uses per-VM virtio serial ports to allow users to copy files, manage process lifecycles, and generally issue any commands that a typical shell supports. The low level of control makes minimega usable for a wide variety of experimental needs, but for repeatedly defining and automating complex tasks, we found it challenging to use these interfaces directly.

This paper presents our work in developing an *experiment description model* (EDM) that provides convenient abstractions with which to rapidly define and automate experiments on minimega. The model consists of a *network* abstraction and a *traffic* abstraction. The network abstraction is largely inspired by existing work in the network emulation community, such as the tcl based definition in ns-2 [14] that allows intuitive ways to define topology, while the traffic abstraction uses a structural format to generate complex mixes of real-world application traffic through a single JSON file. We discuss how these abstractions are encoded by users and how we compile them to the minimega API.

Finally, we showcase the utility of our EDMs by focusing on three case studies in the context of the DARPA Searchlight program [18]. In the first case study, we configure a wide range of topologies to enable evaluation of a distributed network discovery system (Section 4.1). In the second case study, we configure a large number of diverse mixes of application traffic types to enable expansive combinatorial evaluation of a real-time traffic characterization system (Section 4.2). Lastly, in the third case study, we configure a system with complex layer-2 connectivity requirements to enable evaluation of a distributed flow management and traffic engineering system (Section 4.3). In total, we conducted nearly 800 experiment runs to evaluate these three systems during a period of about three weeks.

This paper attempts to capture the lessons learned and key takeaways from our experience of defining and running these experiments on our minimega based network emulation testbed. The minimega framework, albeit with initial challenges, enabled scaling the experiments even when we were restricted by the available physical nodes in the testbed, and provided a set of traffic generators that we augmented with additional tools, as discussed in this paper. We hope that capturing our experience in this paper will make it easier for other experimenters to perform similar experimental evaluations.

2 RELATED WORK

Network Emulation and Simulation. Network emulation testbeds have been fundamental enablers of network and cybersecurity research over the past several decades. Initially created in the earlier 2000s, the Emulab software [32] provides network emulation services on variety of different compute and network devices, and has supported network emulation on a large number of platforms including DeterLAB [34], CloudLab [26], and many others. Part of Emulab's success is based on its flexibility and familiarity to networking and cybersecurity researchers. For example, Emulab users encode network topologies, traffic generators, and other network elements using a tcl based language derived from the well known ns-2 discrete event simulator [14], making it relatively easy to operate Emulab systems for users familiar with simulators. While such simulators, including ns-2, ns-3 [6] and OMNET++ [30] are useful for understanding fundamental network properties, they lack the ability to execute real software, and thus emulation based testbeds have emerged as a popular alternative to enable higher fidelity experimentation that is more generalizable to real world systems.

Network Emulation with Virtualization. In order to further improve scale and flexibility, some testbeds use hardware virtualization to map multiple network elements (including those that require root privileges, such as the Linux kernel network stack), to the same physical nodes. This consolidation of network elements leads to better system utilization, and allows systems to support network topologies with orders of magnitude more virtual nodes than physical ones [22]. This consolidation can be a substantial boon to testbed users as most experiments on emulation-based testbeds use fewer than 3 nodes, a result primarily of resource limitations [12]. The minimega [21] project, on which our work is based, uses Qemu/KVM [3, 16] to emulate nodes and Open vSwitch [23] to emulate network links.

Various other testbeds, including Distem [5], DETERLab [34], Mininet [11], VINI [2], and EmuEdge [35] have used operating-system virtualization techniques, usually called "containers," to functionally isolate emulated network entities on the same node. While these techniques have been shown to reduce latency/jitter that hardware virtualization sometimes

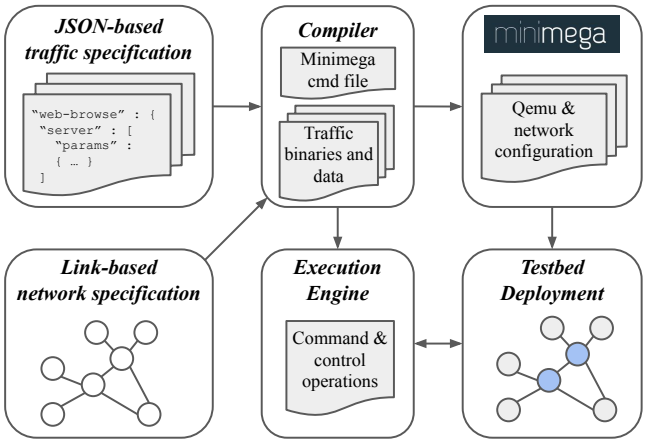


Figure 1: Experiment definition and execution

induces [8], privileged code generally cannot run in containers, which precludes multiple network elements running custom networking stacks (e.g., the Linux kernel or DPDK [24]) from being mapped to the same physical machine. CPU advances have made hardware virtualization overhead small for most operations [17], and in situations where even minor packet jitter impacts an experiment, device passthrough techniques [33] can give VMs direct access to network hardware without sacrificing the functional isolation and privileged environments that VMs provide.

Experiment Description Models and Orchestration. This paper describes our experience using minimega on the DETER testbed [4]. Specifically focusing on how we handled the complexity associated with building varied network and traffic configurations with limited resources on the testbed. Similar efforts have been made to address experimentation challenges on other testbed systems; examples include MAGI [13] for the DETER testbed, OEDL [25] for OMF based testbeds, and Rumba [31] for experiments in GENI and FIRE+ systems. In addition to defining experiment topologies and link emulation criteria, these systems also often seek to enable better experiment orchestration, for example by enabling more graceful handling of errors and providing more control over fine-grained timing and ordering of events. While our experience lead us to focus initially on better support for network and traffic configuration, a possible future extension of our work is to target such an orchestration system in order to provide greater control over runtime behavior.

3 EXPERIMENT DESCRIPTION MODEL

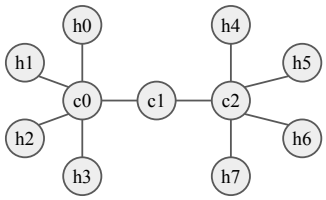
At the outset of the DARPA Searchlight project, our team began the task of evaluating a set of network analysis and traffic engineering projects on a minimega deployed on the DETER testbed. In the early stage of developing basic experiments, we used minimega's mechanisms to manage images, construct simple topologies, control address allocation and routing behavior, and manage the experiment runtime environment, which usually entailed deploying a small number of traffic generators and verifying basic network functionality. For these purposes, we found minemega to be intuitive and mature, and we developed confidence that it could support our experimental requirements. However, as our experimental needs became more complex, we felt a need to develop new mechanisms to enhance and augment minimega. Specifically, we desired new mechanisms to more rapidly define network topologies and application deployment scripts, as well as to emulate a certain class of layer-2 links for which minimega's default link emulation mechanism was insufficient.

This section discusses our experience with addressing these challenges with a structured interface called the *experiment description model* (EDM). Figure 1 illustrates a high level overview of how our system uses EDMs to define and run experiments. The two boxes on the left hand side illustrate the two fundamental pieces of an EDM: a network specification and a traffic specification. A network specification describes the topological properties and other network characteristics of the experiment, while a traffic specification describes the types of applications to deploy on a given network. As shown in Figure 1, the experiment compiler translates a given EDM into a set of command files that configure the testbed's virtualization and command-and-control systems. Sections 3.1 and 3.2 discuss the network and traffic models, respectively, showing how users define them and how the experiment compiler translates them to minimega configurations.

3.1 Network Specification Model

Our network specification model extend's the minimega API by (1) providing a convenient link-centric interface to control the fundamental topological properties of an experiment, and (2) adding a "VM-to-VM" link abstraction which addresses a limitation in minimega's default link emulation strategy.

Link-centric API. Our model extends the minimega API to express topological characteristics through a link-centric API, in a fashion similar to the ns-2 [14] tcl based format common in emulation testbeds. Figure 2 shows an example of how a relatively simple network topology would be programmed in minimega and with our extensions. Figure 2b illustrates the node-centric nature of the minimega API, as each VM encodes a full list of network interface cards (`vm config net`) attached to it. In cases where each link should be considered a point-to-point unicast link, each network name (`foo[0-9]`) must be unique to properly emulate the desired connectivity. We frequently found it difficult to encode



(a) Example topology

```
for i in $(seq 0 7); do
    vm config net foo$i
    vm start h$i
done
vm config net foo0,foo1,foo2,foo3,foo8
vm start c0
vm config net foo4,foo5,foo6,foo7,foo9
vm start c2
vm config net foo8,foo9
vm start c1
```

(b) minimega API

```
for i in $(seq 0 3); do
    connect_vms h$i c0
done
for i in $(seq 4 7); do
    connect_vms h$i c2
done
connect_vms c0 c1
connect_vms c2 c1
```

(c) Extended link based API

Figure 2: Topology specification examples

lists of link names, particularly for complex topologies with large numbers of links on some nodes (e.g., switches and hub routers). Figure 2c shows how we extended the API with a link abstraction that more naturally supports common link definitions. In this way, we felt our API more naturally allowed users to convey topological characteristics, which lead to a less error prone configuration process.

Link types. In addition to standard unicast and multicast links, our model supports a new unicast link type which we call a direct *VM-to-VM* link. VM-to-VM links have the property that traffic sent over them will not pass over any intermediate layer-2 entities, such as switches or bridges, a property that is needed for properly emulating topologies consisting of network elements that may perform their own custom layer-2 forwarding mechanisms (e.g., guest controlled switches). The following section shows how we implement these links, and Section 4.3 demonstrates topologies in which they are required for proper forwarding behavior.

Compilation. The network compilation process translates a network specification to a set of commands to configure minimega and the underlying physical testbed. Our system maintains a list of links for every node in a topology, which is encoded through calls to `connect_vms` in the topology model. At compilation time, the compiler generates a unique VLAN ID for each link and a virtual NIC for each link endpoint, as required by the minimega API. Each virtual NIC has an underlying TAP device configured as a VLAN access port, tagged with the VLAN ID representing that link, and is bridged to an Open vSwitch appliance on the host that transmits packets between the link's endpoints. In situations where the endpoint VMs of a link are mapped to separate physical nodes, we configure virtual tunnel endpoints (VTEPs) with a VXLAN based overlay to transit minimega's traffic across nodes. This is the default minimega approach to link emulation.

Experiments that involve custom layer-2 forwarding mechanisms require a different link emulation strategy. The default approach relies on a MAC address learning strategy to forward packets between the bridge ports, which we found lead to packet loss in at least two situations. The first challenge occurs when packets are forwarded asymmetrically between a source and destination MAC address. For example, if the path from source address X to destination Y passes through the host switch, and the "reverse" path from Y to X does not pass through the switch, then the switch will never actually learn the route to the destination, instead having to broadcast all packets destined to Y through each of its ports, potentially wasting significant memory and network bandwidth in the process. The second challenge arises due to MAC address migration, where a given MAC address X is both the source address and the destination address on separate packets that arrive on a given port of the switch. Address migration typically occurs when VMs are migrated to different hosts, but in one of our use case studies in Section 4.3, a virtualized guest switch is able to generate this behavior through its own custom layer-2 forwarding mechanisms.

We designed direct VM-to-VM links to solve these issues. For these links, we program the switch forwarding databases (FDB) apriori with the specific TAP/VTEP device IDs associated with the link endpoints, thereby allowing packets to be forwarded directly between the two NICs without relying on MAC address learning.

3.2 Traffic Specification Model

The second component of the EDM is a traffic specification, which allows users to define a set of *applications* to run on the network, where applications range from simple file transfers to more complex flows representing common real world applications such as video streaming and web browsing. Our

```
{
  "video-streaming" : {
    "h0" : [{
      "target" : "h4",
      "params" : {
        "client" : {
          "resolution" : "720",
          "protocol" : "hls"
        },
        "server" : {}
  }}]}
}
```

Figure 3: Example traffic model for *Video Streaming*

Table 1: Application types in the traffic specification

Supported Application	Description
File Transfer	File retrieval via FTP(S), HTTP(S), or SCP
IRC	Simple chat messages posted to a common discussion board
Email	Generated email (SMTP) traffic between many hosts
Web Browsing	Randomly generated or specified web pages for website navigation via HTTP(S)
Multi-Client Text Editing	Text editing interaction over SSH
Video Streaming	"One-way" video traffic from a server endpoint to a client endpoint

model is designed to make it easy to define applications by providing a well-defined structural interface that abstracts away details associated with configuring binaries and managing process lifecycles. Furthermore, we sought to allow users to rapidly iterate over diverse sets of applications and to scale to large topologies, without a commensurate increase in the complexity of deploying traffic.

An example of a traffic specification is shown in Figure 3. The user defines a JSON file consisting of a set of *applications*, where each application has a set of *end-hosts* on which it will be deployed. This example shows a single "video-streaming" application with a single client host "h0" and a single server host "h4". The client/server names are human-readable strings that map directly to node names in the network specification. While this example only shows a single client and server, the specification allows the user to encode multiple clients, and each client can target one or multiple servers. Furthermore, the JSON file can encode as many applications as the user desires. The example also shows that parameters ("params") can be encoded in the specification. Parameters are specific to each application type and are passed through to the traffic type's corresponding binary program when it is deployed on the client/server end-hosts. The compilation process has default settings of these parameters for each application type, but parameters give users the opportunity to customize traffic behaviors.

A subset of the applications supported by our system is shown in Table 1. Several of the applications are implemented via protonuke [20], a traffic generator developed as part of the minimega project. This includes the various file transfer variants (FTP(S), SCP, HTTP(S)), email, web browsing and IRC applications. To support two additional real world application types, we developed a multi-client text-editing over SSH application and a video streaming application.

Text editing over SSH. The text editing over SSH application emulates a typical SSH session running a text editing workload. To develop the session, we used minimega's VNC record and replay functionality to record a sample text editing session. The VNC recording can then be played back using six different typing variations to model different speeds and levels of burstiness.

For each SSH application encoded in the traffic model, we first script a command to create an SSH connection between the specified client and server. The VNC session is then replayed on the client from the pre-built recording, using parameters to select speed and burst levels.

Video streaming. The video streaming application serves one-way traffic from the server to the client. The client uses the Google chrome web browser to connect to the server, which hosts a set of video files and supporting website code (HTML, JavaScript). The server runs a Caddy web server [9] with a configuration to serve a static website. The server is currently configured to serve the open source "Big Buck Bunny" video content [27].

To provide a diverse set of video traffic, the server supports multiple video streaming protocols, including dynamic adaptive streaming over HTTP (DASH) [29], HTTP live streaming (HLS) [10], and native HTML5 streaming [1], as well as multiple video resolutions, including 576p, 720p, and 1080p, which can be selected by clients in the traffic specification.

Compilation. The compilation process translates an experiment specification to a set of shell scripts, one for each end-host (traffic generating node) in the network topology. Each end-host thus runs a possibly unique set of programs and arguments for those programs depending on the specification. Compilation entails parsing the JSON file and maintaining, for each unique host, two set of applications, one for which it acts as a client and one as a server, as well as maintaining the parameters passed to the application. Once the file has been parsed, the compiler serializes the startup process for each application by generating a shell script that starts and backgrounds the binaries, as well as a separate script to stop each binary. These scripts are then issued to the minimega

command-and-control system, miniccc [19] at runtime to control the starting and stopping of traffic.

4 CASE STUDIES: DARPA SEARCHLIGHT

This section showcases our experience on evaluating three research technologies developed as part of the DARPA Searchlight program [18]. The Searchlight program seeks novel approaches to analysis and management of an enterprise's distributed applications overlaid on the Internet, with the goal of enabling an enterprise to temporarily decrease the quality of service (QoS) for low-priority application traffic internal to that organization, resulting in sufficient QoS for the organization's high-priority traffic.

The technologies address different components of the Searchlight program goals. Section 4.1 discusses APROPOS, a system for distributed topology discovery; Section 4.2 discusses Fresnel, a system for real-time traffic classification; and Section 4.3 discusses DQM, a traffic engineering system to manage quality-of-service among competing network flows. These following sections specifically capture our experience on the complexities involved in evaluating these technologies, and collectively showcase the technologies required to (1) support a wide range of topologies to evaluate APROPOS, (2) to generate a large variety of network application traffic to evaluate Fresnel, and (3) to support networks with complex layer-2 topological requirements to evaluate DQM.

4.1 Distributed Topology Discovery

APROPOS is a system that attempts to identify network state in real-time. APROPOS uses distributed sensing mechanisms to discover network entities (applications and flows), topological characteristics (location of routers, switches, paths, etc.), and performance characteristics (latency/bandwidth).

Our evaluation of APROPOS focused on its ability to infer topological characteristics on a variety of network topologies. We constructed a set of twelve topologies with sufficient diversity to evaluate APROPOS' different node and path inference mechanisms. Figure 4 shows the four of those twelve that were specifically developed for the task of evaluating the inter-node path sensing mechanism, which infers characteristics of the path(s) between any two APROPOS nodes. Each Figure 4 topology shows three types of nodes: router nodes running APROPOS software (named a^* and shown in blue); generic router nodes (named b^*, c^*, and r^*); and end-hosts at the edges of the network (named h^*). The four topologies in Figure 4 represents tests different functionality: Figure 4a – core with two routers *c[0,1]*; Figure 4b – core with unique central hub router *c*; Figure 4c – core *c[0-4]* with symmetric and/or asymmetric routing; Figure 4d – core *c[0-3]* with equal cost multipath (ECMP) routing.

In each experiment, APROPOS generated a DOT graph showing the topological structure it inferred during the experiment. By comparing these graphs with ground truth information, we determined that APROPOS correctly measured paths for the double dumbbell and star topologies (Figure 4a and 4b). We determined that APROPOS could correctly infer paths when all of the core routers *c[0-4]* in the 5-node loop topology (Figure 4c) were configured to route packets in both directions (symmetric) as well as only in the "clockwise" direction along the loop (asymmetric), but was not yet capable of detecting multi-path ECMP routes in the 4-node loop topology (Figure 4d).

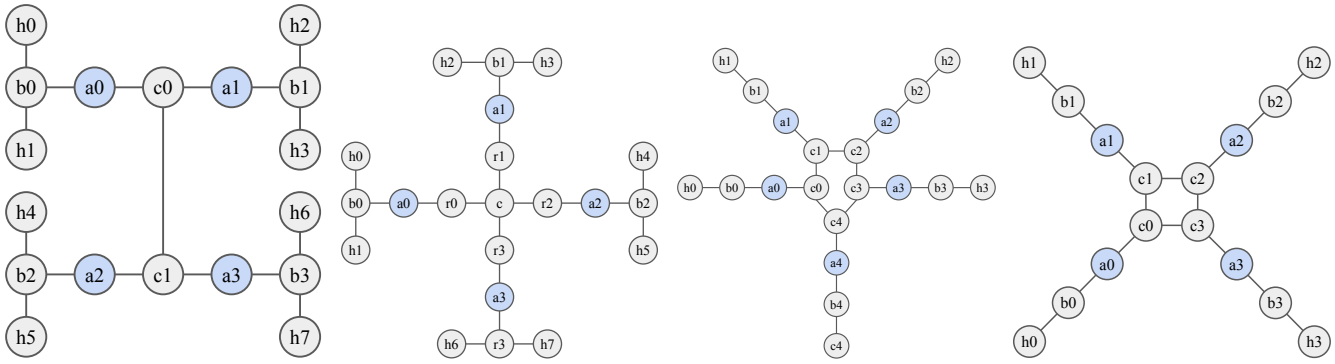
We were able to rapidly conduct this evaluation because these experiments differed only in their respective network definition files. Furthermore, we evaluated several additional inference mechanisms which entailed generating eight additional topologies and routing configurations using our network specification model. We omit detailed discussions of these due to space constraints.

4.2 Real-time Traffic Classification

The Fresnel system identifies traffic flows on the network in real-time. While flow classification is a well addressed problem, Fresnel's goal is to be able to identify flows even in VPN/tunnel encapsulated traffic and also estimate the path(s) that flows are taking. The system has two main processes: a packet handling process and an analysis process. The packet handling process ingests packets captured on the network using DPDK [24] and creates signatures for the flows it sees based on first-order statistics. The analysis then preprocesses signatures, calculates flow attributes moments, and then bins the observed moments to create tensors subsequently used by a clustering algorithm for flow classification.

Fresnel was evaluated on its ability to identify many different application flows and classes in the context of an 18 node dumbbell-like topology, similar to the topology in Figure 4a. There were three types of nodes in the topology: traffic generating end-hosts *h(0-7)*, border routers *b(0-3)*, additional hops in the network *a(0-3)*, and core routers *c(0-1)*. Fresnel was deployed as a layer-2 "bump in the wire" on intermediary layer-2 nodes that sat between each pair of *a* and *b* nodes (not shown in Figure 4a). The system is fully distributed, as the four Fresnel instances in the topology do not communicate with each other.

We developed a range of scenarios to collectively stress Fresnel in a variety of ways, including scaling up the number of concurrent flows to be captured, tagged and clustered. Table 2 lists types of application traffic we used for the scenarios. The first five scenarios evaluate the file transfer classification capability of the system, covering a range of protocols including FTP, FTPS, HTTP, HTTPS, and SCP. The next five



(a) Double dumbbell **(b) Star** **(c) 5-node loop** **(d) 4-node loop**

Figure 4: Subset of the topologies developed for the System A evaluation

Table 2: Applications used to evaluate Fresnel

Traffic Type	Description
FTP	200MB file transfer using the FTP protocol
FTPS	200MB file transfer using the FTPS protocol
HTTP	1GB image transfer using the HTTP protocol
HTTPS	1GB image transfer using the HTTPS protocol
SCP	1GB image transfer using the SCP protocol
text edit slow	emulated user typing at 60 characters/min
text edit medium	emulated user typing at 200 characters/min
text edit fast	emulated user typing at 400 characters/min
text edit bursty	emulated user typing with page up and page down every 100 characters and delay of 1-10 seconds
text edit continuous	emulated user *without* any delays
video DASH	streaming video over the DASH protocol
video HLS	streaming video over the HLS protocol
video HTML5	streaming video via native HTML5
video hi-res	steaming with a high resolution of 1080p
video medium-res	steaming with a medium resolution of 720p
video low-res	steaming with a low resolution of 576p

scenarios evaluate how effective Fresnel is in classifying text editing over SSH and include a range of speeds and user behaviors, namely slow, medium, and fast speeds with bursty or continuous behaviors. Finally, the last six scenarios evaluate Fresnel's classification of video streams over a range of protocols including DASH [29], HLS [10], and HTML5 [1] and resolutions of 1080p, 720p and 576p.

We combined these traffic types to create a total of 26 different combinations of applications to evaluate Fresnel's ability to simultaneously classify multiple distinct traffic classes. In total, we conducted over 500 individual experiment runs for the above 26 different combinations to develop a complete capability and performance characterization. Our traffic specification model was fundamental to enabling the rapid construction of these scenarios, as it internalized the process of managing process lifecycles, constructing and deploying shell scripts for each end-host, and interfacing with the miniccc [19] command-and-control system.

4.3 Distributed Traffic Engineering

DQM is a distributed traffic engineering system that manages how network resources are allocated amongst potentially competing flows in network. DQM interfaces between two separate entities: (1) a high-level operator that specifies application classes and desired QoS metrics for those classes, and (2) a flow identification system, such as APROPOS or Fresnel, that monitors a network and informs DQM about the flows on it. As DQM monitors a network, if it discovers that a Gold flow (highest priority) is not receiving its target bandwidth rate, it determines if any lower priority Bronze flows are competing with it, and if so, reduces the Bronze flows' bandwidth. It does this through a technique called *actuation*, which involves programming an OpenFlow switch to redirect relevant flows through a DQM-controlled node called an *actuator* which then drops a certain percentage of packets to control the flows' bitrates.

Figure 5 shows a topology that was used for one of our DQM evaluation scenarios. In this example, a set of four end-hosts *h{0,1,2,3}* send traffic across a network connected by two border routers *b{0,1}* and one core router *c0*. Two OpenFlow switches (Open vSwitch appliances in minimega VMs) *s[0,1]* connect the border routers and core router, and also have DQC actuator components *d[0,1]* connected via two VM-to-VM links each. DQM programs the flow tables on these switches based on information in the Operator Intent so that, when flows that are Gold/Bronze flow through at runtime and need to have their bitrates adjusted, they can be redirected to *d[0,1]*. Due to the way in which DQM manages these OpenFlow switches, behaviors such as MAC address migration and asymmetric routing may occur, which necessitated the development of VM-to-VM layer 2 link support in minimega, as discussed in Section 3.1.

To evaluate DQM required generating topology and traffic specifications via our EDM framework as well as an Operator

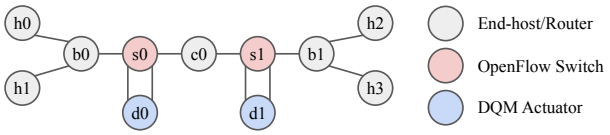


Figure 5: Example evaluation topology for DQM

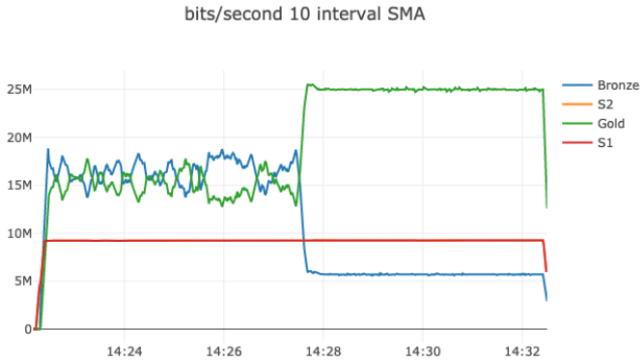


Figure 6: Effects of DQM on competing traffic

Intent to associate each traffic type with an application class and desired bandwidth allocation. An example experiment is illustrated in Figure 6, in which a total of four flows (one Gold, one Bronze, and two Silver) compete for 50 Mbps of bandwidth (all flows have a source in *h[0,1]* and destination in *h[2,3]*. The figure shows 10 second smooth moving average (SMA) of bandwidth utilized by Gold, Bronze, and Silver (S1, S2) flows over the 10 minutes of experiment activity. The figure shows two distinct periods of activity: the first five minutes in which DQM's QoS mechanisms are disabled, and five subsequent minutes starting near the 14:28 mark during which DQM is running. The graph shows that the Bronze and Gold flows, each of which use TCP, initially compete for about 32 Mbps of bandwidth (the remaining 18 Mbps is utilized by the two fixed-rate Silver flows, which each consume 9 Mbps). After actuation, the Bronze flows have their aggregate utilization dropped to a total of about 5 Mbps, while the Gold flows achieve around 25 Mbps, which was the behavior specified by the Operator Intent. This experiment illustrated DQM's successful operation, and showcased our model's ability to support experiments with relatively complex layer-2 topological requirements in the form of VM-to-VM links.

5 TAKEAWAYS

Nuances in link emulation. Minimega's link emulation mechanism entails connecting virtual NICs to a host switch, which uses MAC address learning to forward traffic between the VMs. While this mechanism worked for the majority of our use cases, we encountered situations in our system C evaluation in which more advanced mechanisms were needed, due to the presence of asymmetric routing and MAC address migration. To address these issues, we developed "VM-to-VM" links as discussed in Section 3.1. We suspect that other users would need similar functionality in situations where custom layer-2 forwarding mechanisms are occurring in the VMs. More generally, we found that our link-based network model enabled us to more easily and rapidly generate topologies than the existing VM-based minimega API.

Traffic generation challenges. Creating representative traffic typically requires supporting a wide range of protocols and applications, as well as providing ground-truth and tractability over the traffic generation capabilities. minimega's protonuke provided a good starting point for us, but we needed to augment it in several ways in order to improve traffic usability and realism on our testbed.

We initially struggled to use some protonuke applications that serve content, due to the fact that they required the ability to reach the Internet, which VMs did not have access to by default in our testbed. To address this issue, we configured protonuke to serve from pre-generated content directly. This exemplifies how intended usage models that architects envision may not always match the way in which users use the system, and often occurs when adapting off-the-shelf technologies to new environments.

We developed new applications in order to test more realistic real world traffic such as multi-client text editing and video streaming. This improved our coverage of realistic scenarios and helped improve confidence that the results of our evaluations had some applicability to the real world. Finally, in order to make our experimentation more rapid, we developed a traffic model as part of our EDM that provided a single interface through which to encode desired traffic types, as discussed in Section 3.2. We continue to enhance our traffic generation capabilities, and we believe that having a standardized, centralized interface through which to select and deploy applications will increase adoption.

Managing evaluation process. While the EDM allowed us to more rapidly iterate through experimental scenarios when evaluating the three research prototypes, minimega has several other features that proved useful by themselves. One of the most useful for us was image snapshots. By default, minimega boots VMs on read-only snapshots of their image. This makes it possible to both use the same image for a large number of hosts that should run the same software (e.g., routers), and to prevent accidental image corruption when large numbers of team members are running experiments. In situations where the image needed to be modified, we gave one team member the responsibility to boot the image in read/write mode to make changes.

We made use of several additional minimega features, including its DNS and DHCP servers, traffic shapers for setting link performance constraints, and minirouter for configuring routing protocols in an experiment. We found these features to be able to capture most of our requirements, which obviated the need to make frequent changes to our images.

6 CONCLUSION

This paper captures our experience using minimega [21], an off-the-shelf network emulation system, to support the process of defining and running a large set of complex experiments on the DETER testbed [34] for the DARPA Searrch-Light program. We discussed our approach to enable rapid and less error prone experiment design through an abstraction called the experiment description model. We then illustrated three case studies for network traffic and topology analysis and traffic engineering prototypes. These case studies encompassed several hundred experiments with significant topological and application complexity. We concluded with takeaways describing our experience with minimega more generally, a discussion we believe can generalize to other minimega-based testbeds. We hope our experience will help to enable evaluations of large scale and complex systems using virtualization technologies in the future.

The models, tools and data developed for this paper can be found online at https://mergetb.org/projects/searchlight.

ACKNOWLEDGMENTS

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

REFERENCES

[1] Gary Anthes. HTML5 leads a web revolution. *Communications of the ACM*, 55(7):16–17, 2012.

[2] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures and Protocols for Computer Communications*, (SIGCOMM), 2006.

[3] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In *Proceedings of the Usenix Annual Technical Conference*, (ATC), 2005.

[4] Terry Benzel. The science of cyber security experimentation: The deter project. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, page 137–148, New York, NY, USA, 2011. Association for Computing Machinery.

[5] Tomasz Buchert, Emmanuel Jeanvoine, and Lucas Hussbaum. Emulation at Very Large Scale with Distem. In *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, (CCGRID), 2014.

[6] Lelio Campanile, Marco Gribaudo, Mauro Iacono, Fiammetta Marulli, and Michele Mastroianni. Computer network simulation with ns-3: A systematic literature review. *Electronics*, 9(2), 2020.

[7] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communications Review*, 33(3), 2003.

[8] Jonathan Crussell, Thomas M. Kroeger, Aaron Brown, and Cythnia Phillips. Virtually the Same: Comparing Physical and Virtual Testbeds. In *Proceedings of the 2019 International Conference on Computing, Networking and Communications*, (ICNC), pages 847–853, 2019.

[9] Caddy developers. Caddy - The Ultimate Server. https://caddyserver.com, 2021. [Online: accessed 2020-06-30].

[10] Andrew Fecheyr-Lippens. A Review of HTTP Live Streaming. http://files.andrewsblog.org/http_live_streaming.pdf, 2010. [Online: accessed 2021-02-17].

[11] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible Network Experiments Using Container-Based Emulation. In *Proceedings of the 8th International Conference on emerging Networking EXperiments and Technologies*, (CoNext), 2012.

[12] Fabien Hermenier and Robert Ricci. How to Build a Better Testbed: Lessons from a Decade of Network Experiments on Emulab. In Thanasis Korakis, Michael Zink, and Maximilian Ott, editors, *Testbeds and Research Infrastructure: Development of Networks and Communities*, pages 287–304. Springer, 2012.

[13] Alefiya Hussain, Prateek Jaipuria, Geoff Lawler, Stephen Schwab, and Terry Benzel. Toward Orchestration of Complex Networking Experiments. In *Proceedings of the 13th USENIX Workshop on Cyber Security Experimentation and Test*, (CSET '20), 2020.

[14] Teerawat Issariyakul and Ekram Hossain. *Introduction to network simulator NS2*. Springer, 2012.

[15] Kate Keahey, Joe Mambretti, Paul Ruth, and Dan Stanzione. Chameleon: A Large-Scale, Deeply Reconfigurable Testbed for Computer Science Research. In *Proceedings of the 27th IEEE International Conference on Network Protocols*, (ICNP), 2019.

[16] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the Linux Virtual Machine Monitor. In *The 2007 Ottawa Linux Symposium*, (OLS), 2007.

[17] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. My VM Is Lighter (and Safer) than your Container. In *Proceedings of the 26th ACM Symposium on Operating System Principles*, (SOSP), 2017.

[18] Mr. John-Francis Mergen. Searchlight. https://www.darpa.mil/program/searchlight, 2018. [Online: accessed 2020-06-30].

[19] minimega authors. Command and Control API. https://tip.minimega.org/articles/tutorials/cc.article, 2016. [Online: accessed 2021-02-17].

[20] minimega authors. protonuke simple traffic generation. https://tip.minimega.org/articles/protonuke.article, 2016. [Online: accessed 2021-02-17].

[21] minimega authors. minimega: a distributed vm management tool. https://tip.minimega.org/, 2019. [Online: accessed 2021-02-17].

[22] Ronald Minnich and Don Rudish. Ten Million and One Penguins, or, Lessons Learned from booting millions of virtual machines on HPC systems. In *Proceedings of the 4th Workshop on System-level Virtualization for High Performance Computing*, (HPCVirt), 2010.

[23] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar,

Keith Amidon, and Martin Casado. The Design and Implementation of Open vSwitch. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation*, (NSDI), 2015.

[24] The Linux Foundation Projects. DPDK: Data Plane Development Kit. https://dpdk.org, 2021. [Online: accessed 2021-05-05].

[25] Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. OMF: A Control and Management Framework for Networking Testbeds. *SIGOPS Oper. Syst. Rev.*, 43(4):54–59, 2010.

[26] Robert Ricci and Eric Eide. Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications. *;login: the magazine of USENIX*, 39(6):36–38, 2014.

[27] Ton Roosendaal. Big Buck Bunny. In *Proceedings of the ACM SIGGRAPH ASIA 2008 Computer Animation Festival*, (SIGGRAPH Asia '08), page 62, 2008.

[28] Rusty Russell. Virtio: Towards a de-Facto Standard for Virtual I/O Devices. *SIGOPS Oper. Syst. Rev.*, 42(5):95–103, 2008.

[29] Thomas Stockhammer. Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, (MMSys '11), 2011.

[30] Andras Varga. A Practical Introduction to the OMNeT++ Simulation Framework. In Antonio Virdis and Michael Kirsche, editors, *Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem*, volume 1, pages 3–51. Springer International Publishing, 2019.

[31] Sander Vrijders, Dimitri Staessens, Marco Capitani, and Vincenzo Maffione. Rumba: a Python Framework for Automating Large-Scale Recursive Internet Experiments on GENI and FIRE+. In *Proceedings of the Workshop on Computer and Networking Experimental Research Using Testbeds*, (CNERT '18), 2018.

[32] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2003.

[33] Alex Williamson. VFIO: A user's perspective. In *Proceedings of the KVM Forum*, 2012. [Online: accessed 2021-05-05].

[34] John Wroclawski, Terry Benzel, Jim Blythe, Ted Faber, Alefiya Hussain, Jelena Mirkovic, and Stephen Schwab. DETERLab and the DETER Project. In Rick McGeer, Mark Berman, Chip Elliott, and Robert Ricci, editors, *The GENI Book*, pages 35–62. Springer International Publishing, 2016.

[35] Yukun Zeng, Mengyuan Chao, and Radu Stoleru. EmuEdge: A Hybrid Emulator for Reproducible and Realistic Edge Computing Environments. In *Proceedings of the 2019 IEEE Conference on Fog Computing*, (ICFC), 2019.