Exercises for Graduate Students using GENI

Sriharsha Gangam
Purdue University
Email: sgangam@cs.purdue.edu

Ethan Blanton
Purdue University
Email: eblanton@cs.purdue.edu

Sonia Fahmy
Purdue University
Email: fahmy@cs.purdue.edu

***Abstract*—GENI brings together a wide variety of heterogeneous networking infrastructure and technologies under a common platform. We propose programming exercises for graduate students to introduce GENI and enable students to conduct high fidelity networking experiments. In this paper, we focus on an exercise to study congestion control and reliability using the ProtoGENI aggregate. A planned second exercise aims to leverage GENI OpenFlow aggregates to study firewalls and QoS mechanisms. We believe that these lab exercises will expose students to key networking concepts and recent research directions, *e.g.,* in the data center context.**

I. Introduction

In this paper, we discuss our ongoing work on developing programming exercises for use by instructors of a graduate networking class (*e.g.,* CS 536 in the Computer Science Department at Purdue University). We are designing both: (1) Materials to guide students through the exercises, and (2) Instructor materials, including solutions to the exercises. All exercises require executing experiments on the GENI infrastructure, and understanding the performance that various GENI nodes and links provide.

A key goal of using GENI (as opposed to other platforms such as PlanetLab or DETER) is exploring the diversity available in the GENI infrastructure, and leveraging new GENI projects. GENI provides a common platform to authenticate, describe, advertise and reserve resources for experiments. Students can use a number of tools from the experimenter portal such as Gush, Omni, Raven, and Flack.

The choice of the particular GENI aggregate to use will depend on the goal of each exercise. For example, experiments on the PlanetLab aggregate are useful in predicting path properties over the Internet. The ORBIT aggregate has 400 wireless nodes arranged in a grid. GENI provides a unified management interface for these aggregates and provides opportunities to conduct heterogeneous experiments. Suppose an experimenter needs to study the performance of a Content Distribution Network (CDN)-based live streaming application for wireless customers. The experimenter can reserve (1) ProtoGENI nodes to serve as the CDN and experiment with load balancing/data management, and (2) ORBIT nodes to simulate wireless customers and evaluate the user experience.

Our exercises assume knowledge of the C programming language, basic operating systems concepts including multi-threading, and basic networking concepts. The graduate networking course that will use the planned exercises covers the following material: (1) Network services and applications, including DNS, HTTP, SMTP, and peer-to-peer systems; (2) Network transport architectures, TCP, UDP, and TCP congestion control; (3) Routing and forwarding, including intra-domain and inter-domain routing algorithms; (4) Link layers and local area networks, especially Ethernet and WiFi; and (5) A brief discussion of quality of service (QoS), network measurement and management, and network experimentation and performance analysis. The exercises can also be used in a more advanced graduate networking course.

II. Overview

The planned exercises aim to teach the basics of network programming, client/server architectures, how a reliable transport protocol runs on top of an unreliable delivery mechanism, how routing protocols operate, and how to differentially treat network flows. All exercises require the students to collect, analyze, and explain experimental results from a set of experiments they conduct on GENI.

We have explored a number of ideas for lab exercises. The ideas can be classified into three broad categories:

1) Resource allocation mechanisms,
2) Reliability and congestion control mechanisms, and
3) Packet forwarding, filtering, and quality of service mechanisms.

These correspond to application-layer/middleware, transport-layer, and network-layer functions in the current TCP/IP protocol stack.

In this paper, we will focus on the second category (Section III), but we will also briefly describe ideas for the third category (Section IV).

III. Case Study: Congestion Control

The Transmission Control Protocol (TCP) remains the most popular and widely used transport protocol, contributing the majority of Internet traffic. With the growing data center traffic, and the increasing deployment of high bandwidth links and applications, it is important for TCP to adapt to provide the best service. Recent interest in the performance of Binary Increase Congestion control (BIC) and CUBIC on the end2end-interest mailing list [1], in data center performance [7], [9], [13], [14], and Google's interest in TCP [3] suggest the importance of the topic.

A. Goals

The exercise aims at introducing students to the concepts of reliability and congestion control, and increasing their

```
<?xml version="1.0" encoding="UTF-8"?>
<rspec xmlns="http://www.protogeni.net/resources/rspec/2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.protogeni.net/resources/rspec/2
    http://www.protogeni.net/resources/rspec/2/request.xsd"
    type="request">
    <node client_id="center"
        exclusive="true">
        <sliver_type name="raw-pc">
        <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops//FEDORA10-STD" />
        </sliver_type>
        <interface client_id="center:if0" />
        <interface client_id="center:if1" />
        <interface client_id="center:if2" />
        <interface client_id="center:if3" />
    </node>
    <node client_id="left"
        exclusive="true">
        <sliver_type name="raw-pc">
        <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops//FEDORA10-STD" />
        </sliver_type>
        <interface client_id="left:if0" />
    </node>
    <node client_id="right"
        exclusive="true">
        <sliver_type name="raw-pc">
        <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops//FEDORA10-STD" />
        </sliver_type>
        <interface client_id="right:if0" />
    </node>
    <link client_id="leftLink">
        <interface_ref client_id="left:if0" />
        <interface_ref client_id="center:if0" />
    </link>
    <link client_id="rightLink">
        <interface_ref client_id="right:if0" />
        <interface_ref client_id="center:if1" />
    </link>
    <node client_id="top"
        exclusive="true">
        <sliver_type name="raw-pc">
        <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops//FEDORA10-STD" />
        </sliver_type>
        <interface client_id="top:if0" />
    </node>
    <node client_id="bottom"
        exclusive="true">
        <sliver_type name="raw-pc">
        <disk_image name="urn:publicid:IDN+emulab.net+image+emulab-ops//FEDORA10-STD" />
        </sliver_type>
        <interface client_id="bottom:if0" />
    </node>
    <link client_id="topLink">
        <interface_ref client_id="top:if0" />
        <interface_ref client_id="center:if2" />
    </link>
    <link client_id="bottomLink">
        <interface_ref client_id="bottom:if0" />
        <interface_ref client_id="center:if3" />
    </link>
</rspec>
```

Fig. 1: RSpec request used in the exercise

understanding of the performance of different types of GENI infrastructure. The students will compare TCP behavior and efficiency under different network conditions and TCP parameters. The students can also implement parts of the functions that are typically included at the transport layer in today's TCP/IP stack.

B. Tools Used

This project leverages resources on the ProtoGENI aggregate, using the ProtoGENI test scripts [4]. The ProtoGENI tutorial [5] is a good starting point for students to become familiar with the ProtoGENI aggregate.

Students will also need to familiarize themselves with: (1) **Traffic Control (tc)** available in the GNU Linux distributions on ProtoGENI nodes, found in the `/sbin` directory. In this exercise, `tc` will be used to modify network conditions and enable different scheduling policies. (2) **Iperf [2]** available on the ProtoGENI nodes located at `/usr/local/etc/emulab/emulab-iperf`. Iperf is used to study the performance of TCP. All measurements should be taken at the Iperf server (receiving end).

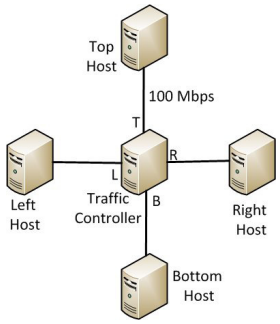

Fig. 2: Star topology of ProtoGENI nodes

C. Reserving ProtoGENI Nodes

After registering a slice using the ProtoGENI test scripts, the student creates a sliver using the RSpec request in Fig. 1. This RSpec represents a sliver containing hosts in a star topology of four nodes connected to a center node with 100 Mbps links as shown in Fig. 2.

The parameters of the interfaces on the "Center" node are modified to mimic different network path properties between the end hosts. Although one could use the delay-node rspec instead, one cannot modify the properties of the delay node once the sliver is created. Further, the delay node functionality is not supported by all GENI aggregates. The TCP parameters and congestion control mechanisms can be varied at the end hosts to compare performance.

D. Tasks

Tasks for the students to complete in this exercise can include any subset of the following:

1) *Impact of Delay/Loss:* This includes basic questions on gathering goodput data and evaluating TCP performance in the presence of delay and loss.
2) *Comparison of Reno and CUBIC:* ProtoGENI nodes provide two TCP congestion control algorithms, CUBIC and Reno, that can be chosen at run-time. CUBIC [10] (and BIC [15]) were proposed for Gigabit links. The CUBIC algorithm is enabled by default. The available algorithms are listed in the file `tcp_available_congestion_control` in the directory `/proc/sys/net/ipv4/`. Congestion control algorithms can be chosen by editing the file `tcp_congestion_control` in the same directory. Based on this setup, we ask the students to compare the goodputs of Reno and CUBIC under different network conditions and traffic. We observed that we need a long TCP Iperf session (∼30 minutes) to notice a performance difference between Reno and CUBIC over 100 Mbps ProtoGENI links.
3) *Study of Queuing Disciplines:* We formulated questions to help the students learn queuing disciplines and understand their utility. For example, we present a scenario with a misbehaving UDP flow in the network and ask the students to demonstrate fairness among the flows

with fair queuing. Similarly, the performance benefits of packet marking schemes such as Explicit Congestion Notification (ECN) can be evaluated under different network and traffic conditions.

4) *Tuning TCP for Lossy Links:* ProtoGENI nodes provide several TCP parameter files in the directory `/proc/sys/net/ipv4/`. The traffic control (tc) program allows users to introduce packet reordering. We ask students to achieve the best TCP performance by learning and tuning TCP knobs. For example, in networks with high re-ordering, one can improve the TCP goodput by adjusting the knobs that help TCP tolerate this re-ordering without assuming packet loss. Similarly, students can answer questions on identifying scenarios (traffic and network conditions) where Selective Acknowledgments (SACK) is most beneficial and verifying them experimentally.
5) *Implementation of Available Bandwidth Estimator:* This programming exercise requires the students to implement a simple available bandwidth estimator based on the notion of packet trains. The overhead, accuracy, and time to calculate the available bandwidth can then be evaluated. Alternately, customized available bandwidth estimators can be developed that work well under specific network conditions (for example, high bandwidth or lossy links).
6) *Implementation of TCP Congestion Control Mechanisms:* Students can partially implement and tune a congestion control module to work well with lossy links or bursty traffic. The students will be provided with a partial TCP implementation to modify and test their congestion control algorithm.

E. Impact of TCP Segmentation Offload on Packet Loss

In developing this exercise, we found that careful examination of the available GENI resources and specification of the RSpec request are necessary. For example, when designing an experiment for the students to observe TCP loss recovery, we found that the network interface cards on some GENI nodes are capable of TCP segmentation offload (TSO), while others are not. TSO is a computation-saving feature that allows the host computer to send IP packets containing very large TCP segments (in this case, the size of about five Ethernet frames) to the hardware, which then divides those large IP packets into frame-size segments for transmission on the wire. This issue arose when hosts were allocated on a different aggregate from that which was used for initial development of the exercise.

The problem with this from a pedagogical standpoint is that TSO is generally disabled during loss recovery. This leads to apparent bizarre behavior when examining TCP streams from the host performing TSO. Fig. 3 presents a TCP time-sequence plot for a host performing TSO (a), and a plot for a host that is not using TSO (b), displaying the unexpected behavior. A TCP time-sequence plot is used to examine the progress of a TCP connection over time. It places time on the horizontal axis, and the TCP sequence space on the vertical axis. Transmitted segments are plotted from the sender, and the cumulative acknowledgment, SACK blocks, and advertised receive window are plotted from the receiver.

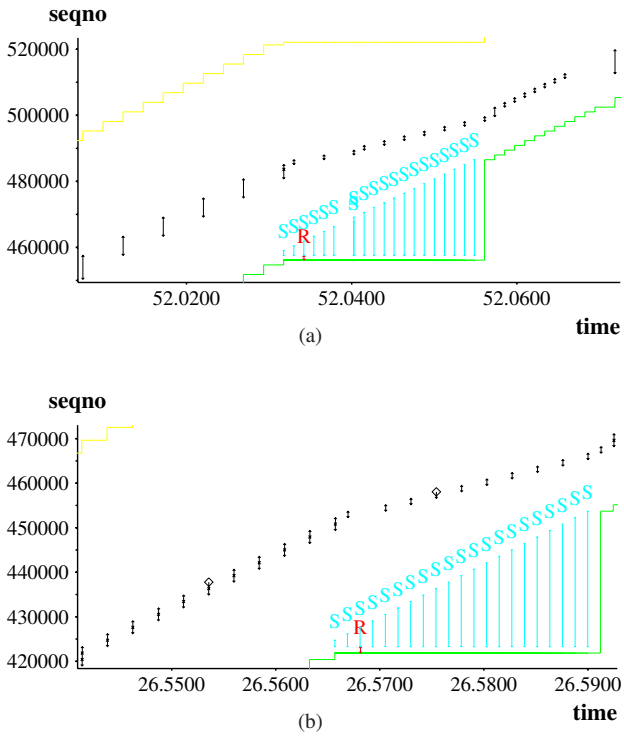

Fig. 3: TCP loss recovery with (a) and without (b) TSO

In Fig. 3, the vertical lines with double arrowheads represent TCP segments as transmitted by the TCP sender. The lower stair-stepped line is the cumulative ACK received from the TCP receiver, and the vertical T-barred lines marked with the letter 'S' are SACK blocks (indicating receipt of a segment or segments higher in the sequence space than the cumulative ACK point). The transmitted segment marked with the letter 'R' is the retransmission of a segment the TCP sender believes was lost (and was in fact lost in this example). The upper stair-stepped line is the receiver's advertised window, and does not come into play in this scenario. Note that the segment that is lost (and subsequently retransmitted) in the plot using TSO appears to be only a small portion of a segment as originally sent. Following this retransmission, the TCP sender appears to greatly reduce its transmitted segment size during loss recovery. While these observations are true from the viewpoint of the TCP sender, they are not true "on the wire," as the segments leaving the TCP sender are repacketized by the network interface and all appear to external hosts to be the same size as the segments transmitted during loss recovery.

TCP segmentation offload, and the impact it has on performance and TCP logic, is an interesting topic in itself. However, when the purpose of an exercise is for students to study TCP loss recovery, the introduction of behaviors such as TSO effects that do not look like "textbook" loss recovery can add unnecessary complexity to the situation. Careful examination

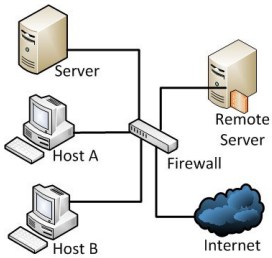

Fig. 4: Experimental setup to implement firewall rules using OpenFlow

of the resources available on the GENI aggregates to be used, coupled with appropriate tweaks to the RSpec request or run-time mitigation (such as disabling TSO on the end hosts, in this example) is necessary.

IV. Future Work: Quality of Service/Filtering Mechanisms

We are currently investigating a second exercise using OpenFlow. OpenFlow provides a centralized control plane management platform. OpenFlow-enabled switches help experimenters define flow rules to support fine-grained flow management. Recently, OpenFlow has been used in traffic and network management proposals in data centers [6], [8], [11], [12]. In this section, we discuss possible programming exercises that introduce OpenFlow to students, and guide them in conducting simple experiments leveraging GENI OpenFlow aggregates.

A. Goals

This exercise can have students use OpenFlow to implement and evaluate different packet filtering and quality of service (QoS) operations on network flows.

B. Tools Used

The exercise can leverage one of the GENI resource allocation tools to simplify requesting resources, and use OpenFlow-capable switches and networks such as the GPO Lab OpenFlow network.

C. Tasks

The students can explore forwarding functionality that OpenFlow-capable switches allow and quantify their performance on multiple aggregates. For example, they can implement a stateful firewall capable of processing established connections without contacting the OpenFlow controller.

Given a network depicted by Fig. 4, the students can develop functionality to implement policies such as: (1) Internet hosts may connect to TCP port 80 on "Server," but no other port. (2) "Server" may connect to "Remote Server" on port 8080, but no other hosts on the right side of the firewall. (3) "Remote Server" may connect to TCP ports 80 or 8080 on "Server", but no other ports. (4) "HostA" and "HostB" may connect to any host on the right side of the firewall on any port, but no host on the right side of the firewall may connect to "HostA" or "HostB."

The controller should examine the first packet of a flow and update flow rules across the switches. All trailing packets are handled by the switches.

The project can be extended to ask students to support efficient traffic engineering functionality or enforce quality of service (QoS) functionality on the flows. This would be especially appropriate for more advanced graduate networking courses.

V. Conclusions

This paper introduces programming exercises to graduate students in a networking course using the GENI infrastructure. GENI enables students to experiment with real networks of a diverse nature. We formulate problems on the evaluation of TCP congestion control and reliability properties, and describe our experiences with TCP segmentation offloading. We also briefly discuss possible exercises using GENI OpenFlow aggregates as part of our future work.

References

[1] The end2end-interest archives. http://mailman.postel.org/pipermail/end2end-interest/2012-February/thread.html.
[2] Iperf. http://iperf.sourceforge.net/.
[3] The official Google code blog. http://googlecode.blogspot.com/2012/01/lets-make-tcp-faster.html.
[4] ProtoGENI test scripts. http://www.protogeni.net/trac/protogeni/wiki/TestScripts.
[5] ProtoGENI tutorial. http://www.protogeni.net/trac/protogeni/wiki/Tutorial.
[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 19–19, 2010.
[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM conference*, pages 63–74, 2010.
[8] T. Benson, A. Akella, A. Shaikh, and S. Sahu. CloudNaaS: a cloud networking platform for enterprise applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 8:1–8:13, 2011.
[9] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 73–82, 2009.
[10] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42:64–74, July 2008.
[11] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 17–17, 2010.
[12] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying NOX to the datacenter. 2011.
[13] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *Proceedings of the ACM SIGCOMM conference*, pages 303–314, 2009.
[14] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM conference*, pages 50–61, 2011.
[15] L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In *IEEE INFOCOM conference*, volume 4, pages 2514–2524, March 2004.