Using GENI to Evaluate Congestion Control Protocols for Next-Generation Networks

Ihsan Ayyub Qazi, Rami Melhem
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260 USA
{ihsan,melhem}@cs.pitt.edu

1. Introduction

The past few years have seen a tremendous increase in the diversity of networked systems, ranging from large-scale data centers to small, highly mobile vehicular networks. These networks have characteristics that differ significantly from the traditional Internet in terms of their capacity, latency, loss behavior, energy requirements, cost, and mobility. This diversity is likely to increase as the Internet evolves to support a much richer set of applications than enabled today by the existing Internet such as remote surgery, 911 access, etc. While such diversity is highly beneficial as it enables new kinds of applications but it also raises new challenges for the design of transport protocols for resource allocation and management. These challenges require that next-generation transport protocols not only achieve better performance but must also provide richer semantics, have evolvable design characteristics, be self-adaptive across diverse networks and have mechanisms for interoperability with other transport protocols.

In order to meet these challenges, there exists an important need to gain experience with new transport protocols and understand the broad implications of their design, many of which can only be assessed with long running, large-scale experiments over heterogenous technologies with real user traffic. Indeed, the history of the Internet informs us that this has been the case with earlier protocols. For example, the need for congestion control was only realized *after* an actual congestion collapse was observed on the Internet [5]. Gaining experience with these protocols will help us in appreciating the real issues and better assess the tradeoffs involved. This will also help us in evolving towards better designs for the future and realize the deployment of next-generation of congestion control protocols

In the last few years, several promising congestion control protocols (e.g., XCP [6], RCP [2], BMCC [13], MaxNet [16]) have been proposed to address performance issues in high-speed networks but their evaluation remains limited to simulations or small testbed experimentations over homogenous technologies [2, 6, 13, 14, 16]. The major reason why these protocols were not tested under larger settings was due to the absence of a large-scale environment for experimentation that allowed programmatically inside the network. Global Environment for Network Innovations (GENI) is such a facility, which aims to support at-scale experimentation on shared, heterogeneous, and highly instrumented infrastructure. In GENI, all components (computers, storage clusters, switches, and sensor networks, etc.,) are expected to be deeply programmable and virtualizable [3, 4]. These characteristics provide an ideal platform for the evaluation of existing and future proposals for congestion control.

In the set of experiments we propose, we to seek to address the following questions related to future transport protocols. We believe these questions will provide the much needed insights that will not only enable future deployments but would also help future designs.

- What level of performance can be achieved from next-generation feedback-based transport protocols in a large-scale environment running over heterogenous technologies with real user traffic?
- How can these experiments inform us about the design characteristics of feedback-based transport protocols in terms of evolvability and self-adaptiveness across diverse networks?
- Can we perform scalable computations inside the network to provide richer congestion feedback to the end-hosts without affecting end-to-end performance?
- How do legacy protocols and technologies impact the performance of new feedback-based end-to-end transport protocols in case the desired congestion feedback cannot be obtained?

In the next section, we outline the resources we need for our experiments, which will then be followed by the experiment design.

2. Resource Requirements

Many proposals for next-generation congestion control protocols require more functionality than provided by existing routers in order to aide end-host decisions. Typically such modifications mean computing a new congestion signal ($e.g.$, load [13], available bandwidth [6], per-flow rates [2]), marking packets based on signal values, and/or treating flows differently [14]. To realize this end, we need a resource which allows programmability inside the routers. We believe OpenFlow [10] and NetFPGAs [9] are suitable resources for our experiments. NetFPGAs would allow scalable processing in the data and control paths, whereas OpenFlow would provide the ability to change routes dynamically and forward traffic to NetFPGAs for scalable processing. At end-systems, we can use a resource such as PlanetLab. We now explain the reasons behind our choices by first providing some background on OpenFlow and PlanetLab.

2.1 OpenFlow and NetFPGA

An OpenFlow switch consists of a *flow table*, which performs packet lookup and forwarding, and has a secure channel to an external controller (possibly located anywhere on the Internet) using the OpenFlow protocol. The flow table contains a set of flow entries (which are a set of header values to match the packet against), activity counters, and a set of zero or more actions to apply to matching packets. A controller is responsible for managing flow table

entries. As of now, the possible header values to match against include ingress port, Ethernet source/destinations addresses, Ethernet type, VLAN id, VLAN priority, IP source/destination addresses, IP protocol, Transport source/destination ports, ICMP Type and ICMP Code. The counters are maintained per-table, per-flow, and per-port. The actions that are required to be supported by all OpenFlow switches include ALL (broadcast to all ports except the incoming port), CONTROLLER (send to controller), LOCAL (send to local networking stack), TABLE (perform actions in flow table), IN_PORT (send the packets out the input port)[1] [10].

OpenFlow switch can forward non-IP packets based on Ethernet addresses or VLAN ids and for processing, these packets can be sent to the OpenFlow controller. Making the controller do processing can be quite useful but in order to provide scalable processing of non-IP packets as well as support periodic computation of congestion signals such as available bandwidth and load, there exist at least two possibilities: (1) OpenFlow switches can route these packets to programmable switches/routers such as NetFPGA-based programmable routers[2] [9] and (2) an OpenFlow implementation on the NetFPGA platform can be used [8]. Note that in the latter case, users would still benefit from the the ability of OpenFlow switches to provide traffic isolation and do dynamic routing [7].

2.1.1 NetFPGA

The NetFPGA is a low-cost reconfigurable hardware platform that is optimized for high-speed networking. The NetFPGA includes the logic resources, memory, and Gigabit Ethernet interfaces necessary to build a complete switch, router, and/or security device. Because the entire datapath is implemented in hardware, the system can support back-to-back packets at full Gigabit line rates and has a processing latency measured in only a few clock cycles. It has a a user programmable Field Programmable Gate Array (FPGA), and four banks of locally-attached Static and Dynamic Random Access Memory (SRAM and DRAM). It also has a standard PCI interface allowing it to be connected to a desktop PC or server [9].

2.2 PlanetLab

PlanetLab provides virtual machines on nodes distributed throughout the world. These nodes are able to virtualize a subset of OS resources and functions such as the CPU, Memory and Storage. They also provide some degree of network stack *isolation*, however, they do not virtualize the network stack (i.e., they do not contextualize the variables in the network stack for each virtual node). As a result, different virtual nodes share a common kernel forwarding table. Virtual machines inside a node share a common IP address and the service provided by these nodes is best-effort in nature.

3. Experimental Protocol

For our experiments, we propose to use the recently proposed Binary Marking Congestion Control (BMCC) protocol that requires explicit feedback from the network [13]. BMCC achieve efficient and fair bandwidth allocations on high bandwidth-delay product networks, while maintaining low persistent queue length and negligible loss rates. Moreover, BMCC reduces average flow completion times by up to 4-5x over TCP and outperforms several other protocols [13].

With BMCC, each router periodically computes the load (ratio of demand to capacity) on its output links and sets appropriate fields in each arriving packet to convey this information to the end-hosts. Note that fields are set in a way such that the maximum load along the path of a flow is communicated to the sources. Based on the load values, sources apply either Multiplicative Increase (MI), Additive Increase (AI) or Multiplicative Decrease (MD) to achieve the desired goals of congestion control (For details, please see the related paper [13]). BMCC requires changes at the end-hosts as well as inside routers.

[1]Actions requiring modifications in the packet fields are optional but it is suggested that at least such VLAN actions be supported.

[2]a typical NetFPGA-based programmable router supports 4 ports

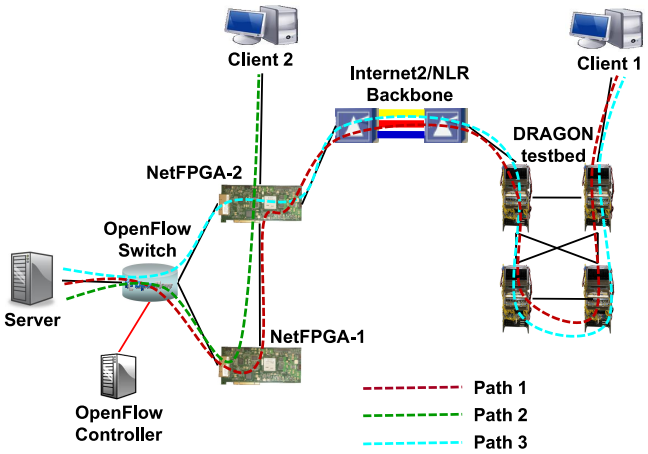

Figure 1: Experimental Setup

3.1 End-systems

BMCC's end-host functionality will be implemented as a loadable kernel module[3]. New congestion control protocols are often implemented as kernel modules to avoid the need to patch the kernel and the subsequent kernel recompilation. The added functionality at the end-hosts will include (i) execution of different sender control laws based on the received load feedback and (ii) setting and interpretation of appropriate packet header fields. The rest of the functionalities required by congestion control protocols such as retransmission timeout estimation and loss-recovery mechanisms will be the same as used by TCP.

3.2 Routers

The BMCC router functionality can be implemented on NetFPGA cards for achieving high performance. NetFPGAs will be programmed to periodically compute load on each of its links. The data plane operations performed by the NetFPGA-based router include updating a byte counter and setting the appropriate fields in the packets' header to convey the router load. The control plane operations take place at a much larger timescale. Link utilization is computed every 200 ms and the queue length is measured every 10 ms and then averaged using a exponentially weighted moving average. Finally, the Load is computed (every 200 ms) as a weighted average of these two values.

4. Experiment Settings

[3]For experimentation with protocols that require changes in the IP header format or require a new header, it is useful to implement this functionality on top of IP (perhaps below TCP) so that traditional routers allow such packets to pass through and only those routers, which are aware of the header, can process them

In the basic experimental setup we consider, there are two clients and one server. The server is connected to an OpenFlow switch which in turn connects to two NetFPGA cards (called NetFPGA-1 and NetFPGA-2) that implement the BMCC router functionality. Client 2 has a direct connection to NetFPGA-2 whereas Client-1's packets first traverse the Internet2/NLR POP and possibly the DRAGON testbed or other infrastructure before reaching NetFPGA-2. Both the NetFPGAs have a direct connection to each other (see Figure 1). We expect to maintain real background traffic on the links, preferably carried from the Internet using solutions such as BGP Mux [15], which provides controlled connectivity to the global Internet routing infrastructure.

4.1 Performance on Large Bandwidth-Delay Product Networks

In the first test, Client 1 downloads a long file via Path 1 as shown in Figure 1. During this transaction, a number of measurements of interest are collected e.g., total file transfer time, time to attain full bottleneck utilization, packet loss rate, etc.,. In the next test, we start two flows with an inter-arrival time of t secs. Flow 1 starts at Client 1 following Path 1. After t secs, a new flow is started at Client 2 which follows Path 2. We measure the time it takes for flow 2 to achieve its fair share[4]. These tests can be repeated to achieve statistically significant results.

4.1.1 Stress Testing

In the next test, we increase the number of users in order stress the underlying infrastructure. We increase the number of client machines and start several flows in each one of them (possibly in the order of 100s if not in 1000s). In addition to the statistics mentioned above, we will also collect other measurements that inform us about the time memory required by the routers to process packets and compute congestion signals, especially under high load. For instance, we plan to profile the CPU usage, which can done using tools such as OProfile [11] in Linux.

4.2 Performance Evaluation under Route Changes

BMCC uses bottleneck load information to adjust its rate. When the flow path changes, the bottleneck may also change, requiring the source to adjust to the new rate[5]. In this scenario, we are interested in understanding how quickly BMCC adapts to the new bottleneck. We start two long-lived BMCC flows as in the previous case. Using the OpenFlow switch controller, we change the Client 1 flow entry to follow Path 3, resulting in a change in the bottleneck.

4.3 Performance Evaluation on a Path with Wired as well as Wireless Segments

In this experiment, we evaluate the performance of BMCC over a network where the last path segment is wireless (comprising of one or several hops). For the last segment, we can use the CMU Wireless Emulator testbed [1] (which is based on NetFPGAs), ORBIT Emulator/field trial network testbed [12], or OpenFlow-enabled Access Points. The advantage of Emulator testbeds is that they allow experimentation for a range of possible wireless scenarios by emulating diverse path characteristics such as in indoor residential environments, outdoor cases, and in mobile settings. Note that this test will require GENI to perform resource reservation across heterogenous wired as well wireless technologies.

[4]For this case, the NetFPGA-1 - NetFPGA-2 link is the bottleneck

[5]This may happen due to client/server mobility, load balancing, or when routers/links fail.

4.4 Performance under a mix of transport protocols

In the test, the goal is to understand the interaction and bandwidth-sharing properties of the protocol. We therefore, start several BMCC as well as several TCP flows simultaneously. We then collect the throughput, loss rate, and response times achieved by flows of the two protocols. This experiment will be run for two cases (a) where the bottleneck is BMCC-enabled and (b) the bottleneck is a drop-tail router.

These are only some of the tests that we plan to conduct using GENI. We will consider other experiments as well as other transport protocols based on the feedback from the community and subject to time limitations.

5. Conclusion

New and diverse networks are emerging at a fast rate. This raises challenges of resource allocation and management in such networks. We need to test new transport protocols using long-running, large-scale experiments across heterogenous technologies before they can be deployed. GENI provides deep programmability and a unique blend of technologies that can enable such experiments for the evaluation of new protocols and architectures. In this paper, we presented a set of experiments to be carried out on GENI and analyzed the resources need for these experiments.

6. References

[1] CMU Wireless Emulator. http://www.cs.cmu.edu/ emulator/.

[2] DUKKIPATI, N., KOBAYASHI, M., ZHANG-SHEN, R., AND MCKEOWN, N. Procesor sharing flows in the internet. In *IWQoS* (Jun 2005).

[3] ELLIOTT, C., AND FALK, A. An update on the geni project. *SIGCOMM Comput. Commun. Rev. 39*, 3 (2009), 28–34.

[4] Global Environment for Network Innovations (GENI). http://www.geni.net/.

[5] JACOBSON, V. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM* (Aug 1988).

[6] KATABI, D., HANDLEY, M., AND ROHRS, C. Internet congestion control for high bandwidth-delay product networks. In *ACM SIGCOMM* (Aug 2002).

[7] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. 38*, 2 (2008), 69–74.

[8] NAOUS, J., ERICKSON, D., COVINGTON, G. A., APPENZELLER, G., AND MCKEOWN, N. Implementing an openflow switch on the netfpga platform. In *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems* (New York, NY, USA, 2008), ACM, pp. 1–9.

[9] NetFPGA. http://netfpga.org/.

[10] The OpenFlow Switch Specification. http://OpenFlowSwtich.org/.

[11] OProfile. http://oprofile.sourceforge.net/news/.

[12] ORBIT. http://www.orbit-lab.org/.

[13] QAZI, I. A., ANDREW, L. L. H., AND ZNATI, T. Congestion control using efficient exlicit feedback. In *IEEE INFOCOM* (Apr 2009).

[14] TAI, C. H., ZHU, J., AND DUKKIPATI, N. Making large scale deployment of RCP practical for real networks. In *IEEE INFOCOM Mini-Symposium* (2008).

[15] VALANCIUS, V., AND FEAMSTER, N. Multiplexing BGP sessions with BGP-Mux. In *CoNEXT '07: Proceedings of the 2007 ACM CoNEXT conference* (New York, NY, USA, 2007), ACM, pp. 1–2.

[16] WYDROWSKI, B., ANDREW, L. L. H., AND ZUKERMAN, M. MaxNet: A congestion control architecture for scalable networks. *IEEE Commun. Lett. 7*, 10 (Oct. 2003), 511–513.