Software Defined Transport: Flexible and Deployable Flow Rate Control

Chi-Yao Hong, Matthew Caesar, P. Brighten Godfrey
University of Illinois at Urbana-Champaign

Background & Motivation: Datacenters are increasingly expected to provide vital support for modern, data-intensive applications such as data-intensive distributed computing, large graph/matrix computation, and online services. The large variety of cloud applications have demanded a diverse range of service requirements such as optimizing completion times [1], meeting task deadlines [6], and satisfying fairness constraints across tenants. However, legacy transport protocols used today, such as TCP, are known to be ill-suited for meeting modern application requirements [6].

To bridge the gap, recent work has focused on optimizing transport protocols for improving flow performance (e.g., D^3 [6], pFabric [1]). Although these protocols offer substantial gains, they have two key problems:

- *Inflexibility*. These protocols have limited flexibility, as each has its own subset of transport policies that it supports (e.g., prioritizing flows or satisfying flow deadlines). It's likely in practice a network operator would like to use particular policies that these systems do not support, such as explicit rate allocation for certain tenants with a mix of prioritization and fair sharing for other flows. [1]
- *Limited deployability*. These protocols have suffered from limited deployability, as most of them require custom modifications to switches, end-hosts, or even both. Consequently, these protocols have seen little practical use, arguably due to their difficulty of deployment.

Software Defined Transport: In this abstract, we propose a *software defined transport* (SDT) architecture for data center transport, in which a central controller computes flow rates and sends instructions to end-hosts. By providing a programmable centralized platform to define transport mechanisms in software, SDT provides greater flexibility. By assigning rates directly to flows at end-hosts, it is substantially more deployable than in-network approaches.

[1] Existing commodity switches typically support only $4-8$ egress QoS classes [6], while a datacenter switch can have several thousand active flows in a one-second time bin [2].

SDT lies in the family of *fabric* architectures [3] which use central control to send instructions to edge devices (in our case, end-hosts or hypervisors), allowing the core of the network to provide only basic packet forwarding functions. But the fabric architecture seems infeasible for fine-grained flow-rate control: given the sub-second timescales needed by rate control, centralized scheduling of thousands of servers' flows would raise a serious latency and scalability concern.

Our goal is to revisit this assumption. How far can we push a fabric architecture towards real-time, fine-grained rate control? To answer this question, we give a preliminary design and evaluation of an optimized SDT architecture. Abstractly, SDT works as follows:

- When an end-host starts a flow, it begins sending using TCP or any other existing transport protocol, and in parallel it sends a demand request to a central transport controller.
- The central transport controller recomputes rates for all the flows in the network and sends new rates to end-hosts periodically.
- End-hosts adjust their rates accordingly.

SDT requires no modifications to today's commodity switches and makes innovation possible as adding a new feature to network simply requires a software update at the transport controller. With the global information about the network traffic and routing, implementing new transport policies in central controller is fairly simple and intuitive. To demonstrate its flexibility, we have implemented a nontrival prototype that supports a rich variety of transport policies such as strict priority scheduling, weighted max-min fairness, or a combination thereof.

Scaling SDT Architecture: Intuitively, only flows that last much longer than the control interval of the transport controller would benefit from this architecture. To make this architecture practical, rate computation has to be fast enough to optimize flow rates in real time at a reasonably large scale. Computing max-min fair rates on one link is trivial, but network-wide, it involves effectively a fluid-flow simulation that has to converge over multiple rounds.

We address the latency issue with two techniques. First, we designed a multi-threaded rate allocation algorithm that emulates link-level queueing disciplines. We found such a multi-threaded design greatly reduces computational time by allowing parallel processing of several links. Second, we handle short, transient flows without the central controller. These flows consume little network resource yet typically require timely delivery. Based upon proper packet header marking (e.g., via DSCP bits) at end-hosts, short flows are initiated with highest queueing priority and do not need to be scheduled by the transport controller until they send more than a certain number of bytes.

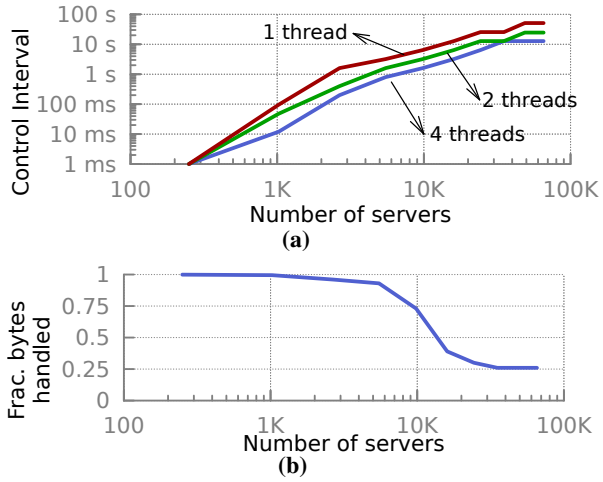

Figure 1: (a,b) Controller scalability test: SDT handles $> 95\%$ of the bytes in a datacenter/cluster of several thousands servers with a control interval of a few hundreds of milliseconds. Our multithreading shows $\approx 4\times$ speedup with 4 threads.

Implementation: We have built a testbed with 112 Linux VMs running on top of VMware vSphere Hypervisor (ESXi), and each VM is equipped with a physical 1-Gbps Ethernet NIC. We have 13 48-port Pronto Switch 3290s running OpenFlow v1.0. Switches are sliced to emulate several smaller switches, so that we can run the testbed as a two-level tree topology: a core switch is connected to 20 top-of-rack switches, each of which has further connected to 4-6 VMs. The transport controller runs on a Dell PC (OptiPlex 9010) with an Intel Core i7-3770K quad-core processor.

We have implemented SDT controller in C++11 using Boost Asio library and Libcurl APIs for control message communication, the host agent in Python with Twisted library. For routing, we use Floodlight OpenFlow controller. The control messages are all sent over TCP. We implement per-flow rate limiting using tc rate-limiter and implement packet marking using iptables in the Linux kernel. A final implementation could be in the hypervisor rather than the OS.

Results: The preliminary results show that our prototype implementation can already scale this architecture reasonably well—a single central server can schedule and dynamically re-schedule flow-rates of 95% of bytes in a cluster with several thousand servers within hundreds of milliseconds (Figure 1). If most bytes belong to short flows, SDT will be ineffective. However, measurements show that, e.g., 95% of bytes belong to flows with duration longer than 10 seconds in a production data center [5], and a similar traffic distribution is observed in many other data centers [2]. For data centers with such traffic, scheduling these flows with a much smaller control interval of hundreds of milliseconds is feasible. This suggests that the centralized rate control architecture is promising for many small and mid-size cloud datacenters or clusters.

Related Work: SDT is close to OpenTCP [4], a SDN-based control layer that dynamically adjusts TCP parameters and variants based on the network traffic measurements. SDT differs from OpenTCP in at least two ways. First, SDT offers greater flexibility than TCP-based congestion control, because the ability to control flow sending rates explicitly allows network operators to implement a wider range of transport policies with SDT. Second, OpenTCP did not show that explicit centralized rate control is feasible. With $\sim$ 4,000 hosts, OpenTCP centrally controlled TCP parameters and variants with a control interval of 60 seconds. In SDT, we show that the more challenging task of explicit network-wide rate control is feasible at a single server with a control interval that is 100x faster than OpenTCP at the same scale.

References

[1] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. Deconstructing datacenter packet transport. In *HotNets*, 2012.

[2] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC*, 2010.

[3] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian. Fabric: a retrospective on evolving SDN. In *HotSDN*, 2012.

[4] M. Ghobadi, S. H. Yeganeh, and Y. Ganjali. Rethinking end-to-end congestion control in software-defined networks. In *HotNets*, 2012.

[5] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *IMC*, 2009.

[6] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron. Better never than late: meeting deadlines in datacenter networks. In *SIGCOMM*, 2011.