eSDN: Rethinking Datacenter Transports Using End-Host SDN Controllers

Hasnain Ali Pirzada, Muhammad Raza Mahboob, Ihsan Ayyub Qazi
Computer Science Department
SBA School of Science and Engineering, LUMS, Pakistan
{15100061,15100140,ihsan.qazi}@lums.edu.pk

ABSTRACT

We propose eSDN; a practical approach for deploying new datacenter transports without requiring any changes to the switches. eSDN uses light-weight SDN controllers at the end-hosts for querying network state. It obviates the need for statistics collection by a centralized controller especially on short timescales. We show that eSDN can scale well and allow a range of datacenter transports to be realized.

CCS Concepts

•**Networks** → **Network architectures; Transport protocols; Data center networks;** *Programmable networks;*

Keywords

SDN; Datacenters; Transport Protocols

1. INTRODUCTION

Modern datacenters host a variety of popular applications such as web search, social networking, and advertising systems. These applications have very demanding performance requirements that critically depend on the underlying transport protocol. As a result, recent proposals focus on designing customized transports for datacenters (DC) to meet a variety of application goals (e.g., low latency) [1, 7]. However, many DC transports either have limited network visibility (e.g., DCTCP [1]) or require changes to switch hardware (e.g., D^3 [7]), which makes their deployment challenging.

Software-Defined Networking (SDN) can offer new opportunities for designing and implementing new DC transports. Existing SDN-based designs rely on centralized controllers for flow setup and collecting network statistics [4]. In contrast to previous approaches, we propose eSDN; a framework that augments centralized controllers with light-weight SDN controllers at the end-hosts. The role of end-host controllers is to periodically query network state (e.g., port utilization and queue length) information that is exposed by SDN/OpenFlow switches. These statistics, along with any application information, can be used to implement a range of DC transports without requiring any changes to the switch hardware. Involving end-hosts is beneficial as they have better visibility into application behavior than network devices, greater computational resources, and more flexibility to adopt new functionality. Moreover, eSDN obviates the need to go to a centralized controller for short timescale operations such as congestion control. This can be particularly beneficial for latency-sensitive short flows.

Previous approaches for obtaining data plane visibility either require custom switch hardware (e.g., Tiny Packet Programs (TPP) [5]) or are too costly at large-scale (e.g., FPGAs and network processors). In addition to being deployment friendly, eSDN offers several other advantages: (a) it can distribute load on a centralized controller to end-host controllers, (b) it enables short timescale operations, and (c) it can allow sharing of information across end-host controllers for better performance. eSDN architecture, however, raises new challenges for scalability and network efficiency. The periodic querying by end-host controllers can lead to high network overhead and requires high switch ASIC-to-controller bandwidth. We analyze the overheads and propose algorithms for reducing them. Moreover, we think that it is likely that future ASICs, designed for use in SDN switches, will provide more bandwidth for this path.

To demonstrate the feasibility of eSDN, we implement an in-network congestion control algorithm (RCP) [3] in ns2. We show how end-hosts, by periodically querying network state, can realize RCP. We would like to point out that for many in-network congestion control algorithms, port or aggregate-flow statistics suffice and more costly per-flow statistics may not be needed. Our initial results show that eSDN can enable high performance for DC transports by using rich data plane information. While eSDN impacts a broad class of network functionality (e.g., network verification), here we focus on the design and evolution of DC transports.

1.1 eSDN Design and Challenges

eSDN's primary goal is to bring low cost network visibility at the end-hosts. To achieve this, each end-host runs a light-weight OpenFlow controller that connects to switches. These controllers run in *slave* mode to ensure that they are read-only (and thus avoid conflicting flow table entries) while saving switch-controller bandwidth as asynchronous switch messages, such as `packet-in`,are not forwarded to *slave* controllers. In addition, each switch is connected to a centralized SDN controller running in master mode that can modify the switch state. The end-host controllers poll switches to collect statistics that can be used by transports.

SIGCOMM '15 August 17-21, 2015, London, United Kingdom

ACM ISBN 978-1-4503-3542-3/15/08.

DOI: http://dx.doi.org/10.1145/2785956.2790022

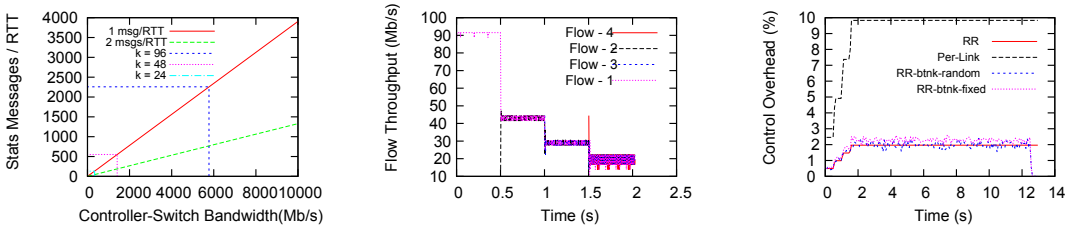

Figure 1: (a) OpenFlow statistics messages as a function of the controller-switch bandwidth. In this scenario, each of the k/2 end-hosts send one or two messages every RTT=200us in a k-ary fat-tree, (b) RCP throughput and (c) control overhead (as a fraction of link capacity=100Mbps) with RCP implementation that uses eSDN.

Switch-Controller Bandwidth: To support high frequency switch state reads, there needs to be enough bandwidth between the switch-ASIC and the controllers (which includes ASIC-CPU bandwidth and the CPU processing capability). Traditionally, switch control processors have been slow e.g., [2] report 17Mbps of switch-controller bandwidth. As ASICs are increasingly being designed for SDN switches (that require greater controller-switch interaction), this bandwidth is increasing [6]. Figure 1(a) shows the number of aggregate Read-State messages that can be handled by a switch as a function of the switch-controller bandwidth for a fat-tree of different sizes. We consider the worst-case intra-rack traffic scenario, where every host sends traffic to every other host. Assuming a polling frequency equal to RTT (=200us), ≈400 requests can be handled by the switch per RTT if the switch-controller bandwidth is 1Gbps.

Network Overhead: eSDN, that polls switches at RTT timescales, can lead to large amount of control traffic. For example, in the previous all-to-all traffic scenario, ≈77.3Mbps network capacity is required for sending 84b (with 44b request sizes and 40b for TCP/IP headers) OpenFlow aggregate-flow stats messages on every link. This amounts to ≈7.7% on a 1Gbps link. For inter-rack scenarios, probing on multiple switches will cause the host-ToR link to consume the most capacity e.g., if a flow traverses through five switches, the access link will incur $5w$ control overhead, where w is the control overhead for one stats message/RTT. This overhead can be reduced by sampling and intelligent polling.

2. CONGESTION CONTROL WITH eSDN

In-Network Transports: In-network transports, such as RCP and D^3, require explicit feedback from the switches. In case of RCP, a router maintains one fair-share rate R per link (of capacity C) which it computes periodically. When the flow count is known accurately, $R = C/N$. Otherwise, it requires knowledge of queue occupancy. With eSDN, end-host controllers can use Openflow's aggregate-flow statistics to get an estimate of the number of flows N. Port stats message can be used to obtain link utilization and queue occupancy. We have implemented eSDN and RCP in ns2. Figure 1(b) shows flow throughput over time for our end-host RCP implementation on a 4-ary fat-tree with 100Mbps links. We ran flows from every host in one pod to a particular host in another pod at different intervals through a particular core link. Thus making a single aggregate-core link the bottleneck. Observe that RCP achieves fair sharing in the face of new arrivals. The polling strategy in this scenario was to poll each link in the flow path every RTT. Due to the polling overhead, flows do not use the entire capacity.

End-host based Transports: Transports, like TCP and DCTCP use implicit (e.g., drops) or coarse-grained feedback (e.g., ECN marks) from the network, respectively. With eSDN, optimal parameters (e.g., initial window size) for these transports can be determined, thus allowing them to achieve better performance.

Algorithms for Reducing Overhead: A naive polling approach at each end-host would be to poll *every* distinct link (per RTT) that is traversed by its flows (termed as `per-link` approach). A better approach would be to poll all links in the flow path in a round-robin fashion (termed as `RR`) i.e., one switch/RTT. While this approach lowers overhead, it can reduce responsiveness. Another approach is to maintain and poll only the *bottleneck* link and re-poll all the switches either after a random or fixed time or after an implicit indication from the network (e.g., packet drops due to change in the bottleneck). Figure 1(c) compares the utilization for a core link with all these approaches for up to 16 flows, each polling for aggregate statistics only. Observe that `RR` approaches can significantly reduce overhead. In RCP, the end-hosts poll for aggregate statistics such as N. Protocols requiring both N and Q would have to fetch both aggregate and port level stats, thereby increasing overhead. An interesting future direction for overhead reduction is to design a peer-to-peer information sharing system so that only one or few controllers within a rack poll for a particular link.

3. CONCLUSION AND FUTURE WORK

We make the following contributions in this paper: (a) We presented eSDN that uses SDN controllers at the end-hosts to realize DC transports, (b) we presented a simple analysis of overhead challenges faced by eSDN and present algorithms for reducing the overhead, and (c) we implemented eSDN and RCP in ns2 and presented an initial evaluation. In the future, we plan to design new schemes for switch load distribution and network overhead reduction.

4. REFERENCES

[1] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *SIGCOMM'10*.

[2] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalag, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *SIGCOMM'11*.

[3] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown. Processor sharing flows in the internet. In *IWQoS*, 2005.

[4] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Software defined transport: Flexible and deployable flow rate control. In *ONS'14*.

[5] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières. Millions of little minions: using packets for low latency network programming and visibility. In *SIGCOMM'14*.

[6] J. C. Mogul and P. Congdon. Hey, you darned counters! get off my asic! In *HotSDN'12*.

[7] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron. Better never than late: Meeting deadlines in datacenter networks. In *SIGCOMM'11*.