Poster: TopoCloud: Getting Datacenter Network Experiments into the Right Shape

Pavani Kuppili, Aleksander Maricq, Brent E. Stephens, Ryan Stutsman, and Robert Ricci
Kahlert School of Computing
University of Utah
USA
pavani.kuppili@utah.edu, aleks.maricq@utah.edu, brent@cs.utah.edu, ryan.stutsman@utah.edu, robert.ricci@utah.edu

***Abstract*—The physical topology of a datacenter network is fundamental as it determines latency, bisection bandwidth, the location and properties of congestion, and the network's ability to tolerate failures. Yet, topology is one of the most difficult factors to control during experimentation: when using a production datacenter or a testbed, the only topology available is the one already deployed. If running on an experimenter's own equipment, rewiring may be possible but is cumbersome and of limited scale.**

This presents a challenge for controlled experimentation, because such experiments should ideally be run on a *range* of realistic topologies. To tackle this, we present our work on Topocloud, a system that enables experimenters to construct and modify network topologies in a shared public testbed. Topocloud uses hardware P4 switches to fulfill a dual role: each port can act as either a typical Ethernet MAC-learning switch or as a "virtual wire" that connects ports transparently. We demonstrate that virtual wires in Topocloud can be used to construct low latency network topologies incorporating L2 switches.

***Index Terms*—datacenter, topology, P4, virtual wire**

I. Introduction

Developing new networked systems and algorithms and driving them to adoption in industry, datacenters, and the Internet requires thorough evaluation under realistic conditions. However, though physical network topology is a key factor in the performance of networked systems, it is generally taken for granted in application-layer works or its impact is only evaluated through simulation or analytical models in works that target the network itself.

We believe it may now be technically feasible to address this problem, in part due to the advent of programmable Layer 2 switches. Our vision is a large public testbed where the network topology is reconfigurable on demand by experimenters. In this testbed, experimental scripts could iterate across different network topologies as easily as they iterate across different numbers of hosts today.

The key questions we hope to answer in designing this testbed are: would the performance of such a testbed be sufficiently realistic to be useful in evaluating new networked systems, algorithms, and topological designs? And, is there a cost-effective way to build such a testbed?

II. Background

Networked applications running in a datacenter environment are significantly influenced by the network topology and tenancy within that datacenter. In distributed computing frameworks like MapReduce [1], the network topology is particularly critical. While the Map stage allows each host to process its data independently, the Reduce stage requires extensive data shuffling across hosts, generating substantial network traffic. Addressing this challenge, some approaches suggest making applications network-topology aware, but these often demand significant modifications to existing software and rely on the assumption that the datacenter's topology is static.

The choice of network topology, therefore, becomes crucial. Traditional Layer 2 topologies [2] facilitate forwarding decisions based on packet data or metadata, which may introduce latency overhead when the forwarding paths are static, effectively functioning as "virtual wires" between ports. Layer 1 topologies [3], on the other hand, offer lower latency forwarding but lack the ability to dynamically switch based on packet content. Programmable switch elements offer a balanced solution by enabling low-latency forwarding combined with reconfigurability. For instance, in topologically reconfigurable testbeds, programmable switches not only help in constructing adaptable network topologies but also provide experimenters the ability to leverage these programmable elements to explore new network functionalities, serving a dual purpose in both topology management and experimental flexibility.

III. Approach

In our research, we focus on designing and configuring a network topology that incorporates L2 switches. Traditionally, these L2 switches are interconnected using L1 switches. To emulate this setup in a software-defined environment, we utilize two P4 programs on an Edgecore DCS800 Tofino switch [4]: (i) an L2 MAC-learning switch [5], and (ii) a "virtual wire" mechanism to interconnect the L2 switches.

We configured the Tofino switch to operate as an Ethernet MAC-learning switch between CloudLab servers [6]. The switch is programmed with two key match-action tables: the arp_table, which processes ARP requests by executing a broadcast action, and the forward_table, which handles forwarding decisions for ARP replies by matching on destination IP addresses.

For the "virtual wire", we implemented packet forwarding logic on the Tofino switch with one key match-action table.

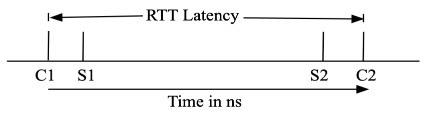
This table operates on the ingress pipeline and uses the input packet's ingress port id (from the port connected to the source server) as its key. Upon an exact key match, the table executes the forward action to the egress port id connected to the destination server specified in the P4 code or the control plane.

Our experiment setup has two c6525-100g nodes [7] connected over our Tofino switch, a Dell S5248 layer 2 switch, and connected over a direct attach cable (DAC). These nodes each have two NICs; we exclusively used the Dual-port Mellanox ConnectX-5 Ex 100Gb NIC in each node for our experiment.

Tofino switches support cut-through forwarding in both unicast and multicast scenarios. While cut-through forwarding is enabled by default for unicast traffic (with the option to disable it), it must be explicitly enabled for multicast forwarding within the P4 program. In our experiments, we have enabled multicast cut-through forwarding.

We used a modified version of the SLA-NG probed [8] tool to measure the round-trip time (RTT) latency between these connections. We ping with one tcp packet at a time with a wait time of 1ms between PINGs from client server. Rather than incur the overhead of timestamps taken in the kernel, the tool utilizes hardware timestamps on the NIC to take RTT measurements as close to wire-time latency as possible. To better understand how RTT is calculated by SLA-NG's probed, we can refer to Figure 1. C1/S2 are taken as the packet leaves the NIC, and S1/C2 are taken as the packet is received by the NIC. At timestamp C1, the client sends a packet to the server. The server receives the packet at timestamp S1. After response generation, the server sends the response packet back to the client at timestamp S2. The client receives the response packet at timestamp C2. The client calculates the RTT as (C2 - C1) - (S2 - S1).

The question we are addressing is whether we can build a lower latency connection using virtual wire or Tofino as an Ethernet switch? Our measurements say "yes".



C1	Client sends packet to server	S2	Server sends packet back to client
S1	Server receives packet	C2	Client receives packet

Fig. 1. RTT latency measured by SLA-NG probed

IV. Results

Using Tofino as a virtual wire between two nodes increases the RTT latency by 182-217% compared to a DAC. However, it decreases RTT latency by 17-20% compared to an L2 switch. We recorded 1000 data points for each RTT latency measurement and plotted CDFs, as shown in Figure 2.

The RTT latency data for DC has a median value of 455 ns. For an L2 connection the data has a median of 1703 ns. The median value for RTT latency data over a Tofino (working as either "virtual wire" or Ethernet Switch) switch is 1446 ns. We observe a significant reduction in RTT latency using the Tofino switch with respect to an L2 connection. Using Tofino as an Ethernet Switch yields similar results even when accounting for the MAC-learning process. While the Tofino switch's performance is not as low-latency as a DC connection, it demonstrates a promising balance between reconfigurability and overhead. The Tofino switch can act as both an L1 switch for partitioning and reconfiguring experiments and as an L2 switch for forwarding, potentially allowing us to achieve flexible and efficient network topologies. This capability suggests that the Tofino switch could enable dynamic reconfiguration of network topologies with minimal latency overhead, making it a viable solution for environments where both low latency and reconfigurability are crucial.

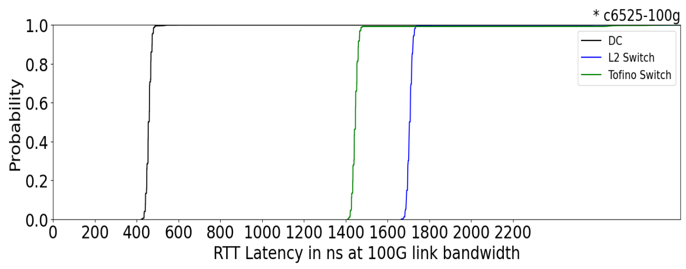


Fig. 2. CDF plot for latency measurements for c6525-100g nodes at 100Gb bandwidth

V. Future Work

Our future work will include: (a) using re-circulation to enable more complex topologies without requiring many physical ports; and (b) building a control plane to dynamically re-configure Topocloud to replicate arbitrary topologies.

References

[1] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[2] Todd Lammle. Layer 2 Switching, pages 193–218. 01 2020.

[3] Turner, Jonathan. (1986). New directions in communications (or Which way to the information age). Communications Magazine, IEEE. 24. 8-15. 10.1109/MCOM.1986.1092946.

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: programming protocol-independent packet processors. SIGCOMM Comput. Commun. Rev. 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[5] Spurgeon, C. E., & Zimmerman, J. (2014). Ethernet: The Definitive Guide. O'Reilly Media.

[6] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The design and operation of cloudlab. In Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '19). USENIX Association, USA, 1–14.

[7] CloudLab. Cloudlab hardware. https://docs.cloudlab.us/ hardware.html.

[8] Amaricq. Slang-probed. https://gitlab.flux.utah.edu/amaricq/ SLANG-probed.