Dolphin: Dataplane Load-balancing in Programmable Hybrid Networks

Andrew Or, Matheus V. X. Ferreira, Chaitanya Aluru

Abstract

Programmability allows networks to adopt flexible policies without requiring hardware changes every time a new policy is deployed. However, the first step to transition to such a network today is to first replace all existing devices with programmable ones. This is often a major hurdle for enterprises to upgrade due to the high upfront operational costs and inherent risks involved.

In this paper, we explore how to deploy programmable networks incrementally, such that networks can still reap the benefits of programmability by upgrading only a small fraction of their switches. Our work targets the use case of load-balancing at dataplane time scales. Existing schemes, such as HULA [9] and CONGA [3], require devices in the network to be homogeneous and capable of functionality not available on legacy switches. We propose two schemes to approximate the same behavior on hybrid networks where only a fraction of the switches are programmable.

1. INTRODUCTION

The rise of software-defined networks is a testament to the importance of flexible network policies. Existing deployments have, in large part, adopted OpenFlow [12] as the standard protocol to communicate routing state between the controllers and the switches in the network. Over time, however, increasing adoption has revealed that this fixed header match-action paradigm is not flexible enough for many use cases, prompting researchers to explore alternatives that provide a higher degree of programmability in the network.

Recent work has illustrated how programmability in the network can be leveraged to build new, flexible algorithms not possible with the OpenFlow paradigm. An example of this is HULA [9], a datacenter load-balancing algorithm that operates at dataplane time scales. The system sends periodic probes into the network and computes routes based on the congestion information reported by these probes. The algorithm is implemented using P4 [4], a high-level language for specifying switch behavior.

Today's datacenters typically employ multi-rooted topologies that use a large degree of multipathing, which provides the network with opportunities to distribute load across different paths by splitting flows. Before HULA, the canonical way to do this is ECMP [6], which decides the path a given flow will take by hashing on the flow's five tuples. A crucial limitation of ECMP is that, unlike its successors, it is oblivious to congestion. This leads to poor load-balancing when the network load increases or when the topology becomes asymmetric due to link failures. Fine-grained load-balancing schemes like HULA are important because reacting quickly to congestion can lead to drastic performance improvements.

One requirement for HULA is device homogeneity. This prerequisite is shabbier by another load-balancing scheme, CONGA [3], that HULA strived to outperform. In particular, HULA assumes that all switches in the network are P4 switches while CONGA requires custom hardware. In either case, however, the heavy operational burden involved in upgrading all the devices in a network makes deploying these applications difficult. It is not uncommon for modern data centers to have tens of thousands of machines. Upgrading all of these at once may be prohibitively expensive and operationally risky.

In this paper, we explore the extent to which the benefits of dataplane time scale load-balancing can still be realized by upgrading only a small fraction of the network. We make the following contributions:

- We discuss the inherent challenges involved in designing a load-balancing scheme for hybrid networks (§2)
- We propose two schemes targeting such networks and discuss the design choices involved (§3, §4)
- We evaluate the effectiveness of these schemes by simulating them in heavily loaded networks and comparing them against HULA (§5)

2. DISCUSSION

The key challenges in designing a load-balancing scheme for hybrid networks are two-fold. First, we no longer have access to fine-grained link statistics, used extensively by HULA and CONGA to make load-balancing decisions, at every hop of the network. The tradeoff here is that upgrading fewer switches, though more conducive to actual deployment, leads to fewer opportunities to make load-balancing decisions based on fine-grained statistics collected from the network core.

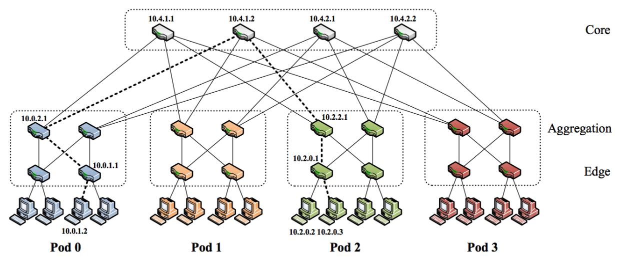

Figure 1: k-pod fat tree [2]

This makes the location of the programmable switches in the network an important choice.

The second key challenge is reconciling differences in forwarding behavior between the programmable switches and the legacy switches. A side effect of hybrid networks is that introducing custom routing behavior at the programmable switches inevitably gives rise to two different control planes running in the same network. In this setting, even if the programmable switches have perfect vision into all utilization and latency statistics in the network, they must still either (1) take into account the forwarding behavior of legacy switches when making their own forwarding decisions, or (2) communicate the information they collected to the legacy switches somehow. The former is cumbersome to reason about and may lead to inflated paths. The latter requires the two kinds of switches in the network to speak the same protocol, which may not be easy since the legacy switches are very limited in this regard.

Beyond these two challenges, an important observation is that it is actually impossible to achieve true, HULA-style dataplane time scales in a network where a significant fraction of devices are legacy switches. This is because the only way to modify forwarding state in legacy switches is through static configurations or control messages such as routing advertisements. Unlike programmable switches, legacy switches are not capable of updating their forwarding tables while processing data plane packets. Even if the hybrid network is comprised of OpenFlow switches instead of legacy switches, modifications to forwarding tables at the OpenFlow switches must still go through the control plane. Thus, any load-balancing scheme designed for hybrid networks at best approximates true dataplane time scales in run time. In other words, the question to answer is not whether the scheme runs in dataplane time scales, but rather how close it is.

This paper proposes two different load-balancing schemes for hybrid networks. We target datacenter settings and assume a k-pod fat tree topology as described in [2] for ease of explanation, though they can be applied to general multi-rooted topologies.

Latency-sensitive scheme. Programmable switches are installed at the core of the network. Periodically, each host exchanges probes with all core switches. From these probes, the core switches learn the round trip delays to each host and use these delays to make forwarding decisions at the core layer. They then propagate this latency information to legacy switches in the aggregation and leaf layers in the form of routing advertisements, announcing higher costs to destinations whose probes faced higher delays. This will be described in §3.

Utilization-sensitive scheme. Programmable switches are installed at the aggregation level of the tree. Hosts periodically send probes to each other. Upon receiving the probes, programmable switches modify the header with local link statistics as in HULA while legacy switches may set the ECN flag to indicate congestion. Programmable switches then make forwarding decisions based on both the link utilization information reported by other programmable switches and the presence (or absence) of the ECN bit. This will be described in §4.

It is worth noting that the programmable switch placement described here is just the starting point. The expectation is that as more switches in the network are upgraded, the load-balancing schemes will be able to make more informed forwarding decisions. In both schemes, the extreme of upgrading all switches to programmable ones results in something that resembles HULA. In the latency-sensitive scheme, each hop will learn the latency to each host, obviating the need for core switches to propagate this information through routing advertisements. In the utilization-sensitive scheme, each hop will have access to local link utilization statistics, obviating the need to indicate congestion through the coarse-grained, utilization-unaware ECN flag.

3. LATENCY-SENSITIVE SCHEME

In this section, we describe a scheme that makes load-balancing decisions based on probe latencies. We assume the setting where the programmable switches are P4 switches and the rest of the switches in the network are legacy ones running some shortest paths protocol with ECMP. Hosts are attached to leaf switches at the edge and periodically exchange probes with the programmable switches in the network core.

3.1 Programmable switch placement

As a starting point, we chose to install the programmable switches in the core layer of the tree. There are a few desirable properties that result from this decision:

1. **All inter-pod traffic traverses a programmable switch.** Placing the programmable switches at the core of the network maximizes the number of paths that will traverse at least one programmable switch while minimizing the number of switches to upgrade. More specifically, all inter-pod paths traverse exactly one programmable switch, thus leaving the network with an opportunity to load-balance flows for a majority of the paths.

2. **Exactly one path between a core switch and a host.** Since probes from each host has only one way get to

a particular core switch, the round trip delays reported by these probes always describe the same path. This simplifies collecting host latencies significantly.

3. **Only 20% of all switches need to be upgraded.** In a k-pod fat tree where all switches have k ports, there are $(k/2)^2$ core switches, $k^2/2$ aggregation switches, and $k^2/2$ leaf switches [2]. Thus, the fraction of core switches in the network is

$$\frac{(k/2)^2}{(k/2)^2 + k^2/2 + k^2/2} = \frac{\frac{1}{4}k^2}{k^2(\frac{1}{4} + \frac{1}{2} + \frac{1}{2})} = 1/5$$

Note that this placement of programmable switches is only a starting point. Switches in the aggregation and leaf layers of the network can be upgraded in the future to achieve finer-grained load-balancing if necessary. In particular, under the current scheme where only the core switches are programmable, intra-pod traffic suffers from the same limitations faced by ECMP because the core switches are unaware of the choice of paths available to these flows. Upgrading even just the aggregation layer can solve this problem since the programmable switches will now be aware of the round trip delays between the aggregation switches and the hosts.

3.2 Measuring host latencies

Core switches learn the latency to each host through the round trip delays recorded by probes sent by the host. This information is used as a proxy of congestion in the network. If there exist significant queues on a path to a given host, then we should expect probes sent on the path to experience noticeable delay. Because probes are sent in the dataplane, the delay they experience also affects real datacenter traffic.

One complication in measuring host latencies this way is the presence of bidirectional links. Delays experienced in one way may not match delays experienced in the other because the queues may be present in only one direction. When hosts and programmable switches exchange probes, there is no way to differentiate between the delay experienced by the probe in one direction versus the other. This is not an issue, however, because most datacenter traffic, just like the probes, requires a response from the destination. Thus, even if a path is congested in only one direction, either the request or the response will experience some delay.

Another subtlety is that clocks may not be synchronized across devices in the network. This means the same machine must be responsible for recording the timestamps. In our case, the host is a more suitable location for originating packets since P4 switches are packet parsers. After a round trip, then, the host must then inform the core switch of the round trip delay through another probe. Thus, the probe exchange between a host and a core switch is actually a three-way exchange. This extra cost may be alleviated in the future as improved precision in datacenter clock synchronization [11] renders an entire round trip in this exchange unnecessary.

3.3 Propagating news of congestion

The flip side of having exactly one path between any given host and core switch (property 2 in §3.1) is that the programmable switches can't be the agents in the network that actually split the flows. This is because the core switches not have an alternative path to choose from in case the one and only path to the host is congested. Thus, although the load-balancing decisions originate from the core switches, they must be propagated to the aggregation and leaf switches, which have many more path choices.

To propagate congestion information, programmable switches must speak the same protocol as the legacy switches. In our scheme, we assume legacy switches run a shortest paths routing protocol, so the programmable switches can embed the latency to a given host in a routing advertisement for that host. Once again, the decision of placing the programmable switches in the core layer has the effect of simplifying the programmable switch logic. Under this placement, each pod essentially becomes an island whose boundaries are marked by the core switches. Thus, regular shortest paths is only used within each island while programmable switches make congestion-aware decisions at the higher level, even though the entire network runs the same routing protocol under the hood.

Simply propagating the destination host latency turns out to be insufficient, however. Doing so only considers congestion on the way down, but not on the way up. More specifically, even if the path between the source host and the core switch is congested, this scheme as it is will just happily route packets on this path as long as the path between the core switch and the destination host is not congested.

Thus, the core switches need to also capture the source host latency somehow when advertising to legacy switches. Note that we can't simply advertise the source host latency, however, because standard routing advertisements are destination based. Here we observe that congestion between an aggregation switch and a core switch is likely if any of the links downstream of the aggregation switch are congested, especially if the datacenter is oversubscribed. Thus, the core switch also keeps track of, for each aggregation switch, the average latency of all downstream hosts and uses this in the routing advertisement to that switch.

In summary, the routing cost core switches advertise to downstream switches is the sum of (1) the latency to the destination host, (2) the average latency of all hosts downstream of the egress port, and (3) some base constant (a large number) for guaranteeing that these fake advertisements do not interfere with regular shortest paths routing.

4. UTILIZATION-AWARE SCHEME

In this section, we describe a scheme where OpenFlow switches interact with P4 switches executing HULA load-balancing protocol. OpenFlow [12] similarly to P4 is part of the SDN (Software Defined Networking) movement, providing many of the flexibilities that a programmable switch can provide with the advantage of already enjoy large-scale deployment in the real world [7]. While P4 aim for hardware programmability, OpenFlow offload all control plane to a logically centralized controller.

For the integration of legacy switches with P4, we should provide the correct loop-free broadcast of probes. Such mechanism can be implemented with multicast groups. Firstly, on a tree topology we can define a path from a core switch to a leaf switch (TOR). For a given switch s, an upstream switch u is a switch reachable from a loop-free path starting at s and ending at core switch c. Similarly, for a switch s, a downstream switch d is a switch reachable from a loop-free path starting at s and ending at a leaf switch l. For a switch, an upstream port receives/sends traffic from/to an upstream switch, and a downstream port receives/sends traffic from/to a downstream switch. This port classification is disjoint; a port can either be a downstream port or an upstream port but not both. We define two multicast groups: (1) when a probe enters a downstream port we multicast to all ports; (2) when a probe enters a upstream port we multicast to downstream ports only. For P4 switches, the multicast groups are pre-installed and the selection of the multicast group for each port is inserted on P4 tables.

Since we assume we cannot retrieve utilization information from legacy switches, a P4 switch can receive probes advertising paths with higher utilization than what actually is provided. A possible work around is with the use of ECN (Explicit Congestion Notification) flags to identify possible congested paths.

For the load balancing with OpenFlow switches, we implemented ECMP by hashing in the packets 5 tuple. We let for future work to combine the routing algorithm described in 3 with OpenFlow switches. We extended the probes to include flag bits. The downstream bit is set to 1 by a P4 switch in the egress when a probe is being forwarded to a downstream port. This flag allows the OpenFlow controller to learn the topology from probes and setup its multicast groups.

4.1 Implementation

The scheme was simulated with Mininet [10], using Open vSwitch and the P4 reference switch. We implemented our SDN controller with POX. Mininet was extended to support networks with hybrid switches by providing a class wrapper as our network switch. The class wrapper uses python reflection to forward functions calls and field accesses to either `OVSSwitch` or our `DolphinSwitch`. In addition, we extended the P4 Mininet module to provide custom options to our reference switch. Similarly to HULA, it was necessary to extend the P4 specification, modifying the reference switch to provide utilization information in ingress and egress ports. In addition, we instrument the reference switch to get measurements.

Since the current P4 reference switch provides limited performance, to fairly compare the performance of our scheme combining Open vSwitches and the reference switch, we limit link bandwidths to 5Mbits/s. For a path going through five P4 switches, we have a RTT of 10ms; therefore, we set probes to be sent in a period of 40ms. We use AF_UNIX sockets to collect measurement between the processes.

5. EVALUATION

5.1 Latency-sensitive scheme

We implemented a simulation of the latency-sensitive scheme described in §3 in ns2. Our evaluation uses a 4-pod fat tree where each top of rack switch has 8 hosts attached. The links between the hosts and the top of rack switches are 100Mbps while the rest of the links in the network are 400Mbps.

In our workload, all the hosts in two of the pods send large files over FTP over TCP to hosts in the other two pods. To simulate varying load in the network, we had the same hosts send a constant rate of traffic over UDP to the same destinations at the same time.

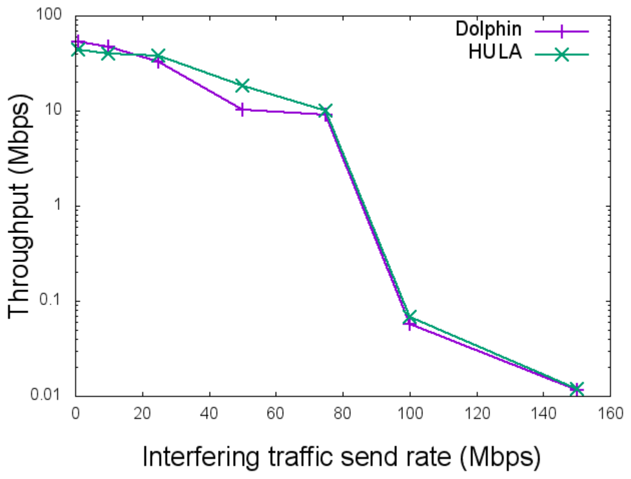

Figure 2: Dolphin vs HULA throughput with inter-pod traffic only

Figure 1 shows how Dolphin, our latency-sensitive scheme, compares against HULA under this workload. We are concerned with the throughput of the FTP applications. For most load, throughput in Dolphin tracks that in HULA fairly closely. Under medium load, furthest Dolphin ever deviated from HULA was by 43.2% when interfering sending rate was at 50Mbps. Under high load, when the file transfers are still making progress, Dolphin deviates from HULA only by 8.6%. As the network load increases further, throughputs in both schemes drop together.

Note that the above workload comprises of only inter-pod traffic. As we discussed in §3, the programmable switches are not aware of host latencies within the pod and thus do not load-balance intra-pod traffic, leaving it to standard ECMP. Thus, we simulated another workload that adds intra-pod traffic to the above workload and expected the throughput to degrade relative to HULA.

Figure 2 shows how Dolphin compares against HULA when there exists intra-pod as well as inter-pod traffic. Compared

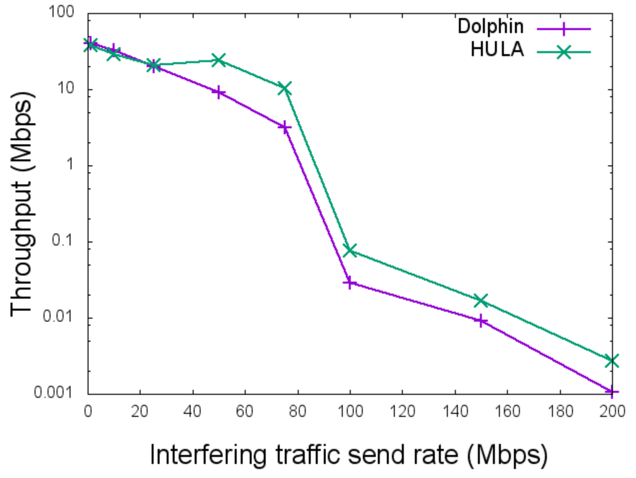

Figure 3: Dolphin vs HULA throughput with both intra-pod and inter-pod traffic

to the previous inter-pod only workload, Dolphin deviates from HULA by wider margins even under high load. In particular, under medium load, throughput in Dolphin deviates from that in HULA by 61.7%. Under high load, Dolphin deviates from HULA by as much as 70%. This highlights the shortcoming of load-balancing only inter-pod traffic and leaving ECMP to handle the rest.

5.2 Utilization-sensitive scheme

To evaluate under our Mininet testbed we used a Fat-Tree topology with 5Mbits/s links, 2 pods with 2 TOR switches and 2 aggregation switches in each pod and 4 core switches connecting the two pods. The topology can provide a maximum capacity of $S = 20$Mbits/s. We use a methodology similar to Hula. Each TOR switch connects to a host running 8 servers listening to different TCP ports and one client. Each client start 3 connections with 3 random server in other TORs.

To generate flows, we draw flow sizes from a gaussian distribution with mean $\mu = 30$Kbytes and standard deviation of $\sigma = 10$Kbytes. We vary the load in the network by varying the mean λ of a poisson distribution where we draw the flow inter-arrival time. For each flow sent, we measure its FCT (Flow Completion Time). We run the tests twice, (1) one when the topology is completely composed by P4 switches and (2) when the network has P4 switches only as the aggregation switches.

Considering the average flow size μ, the average flow inter-arrival time λ, the number of connections C, and the network capacity S, we can compute the expected network load as,

$$L = \frac{\mu}{\lambda}\frac{C}{S}$$

In Figure 4, we see the average FCT for all flows when we vary the load in the network. We obtain partial results because the software OpenFlow, Open vSwitch, provides better performance than the P4 reference switch that targets performance when implemented in hardware.

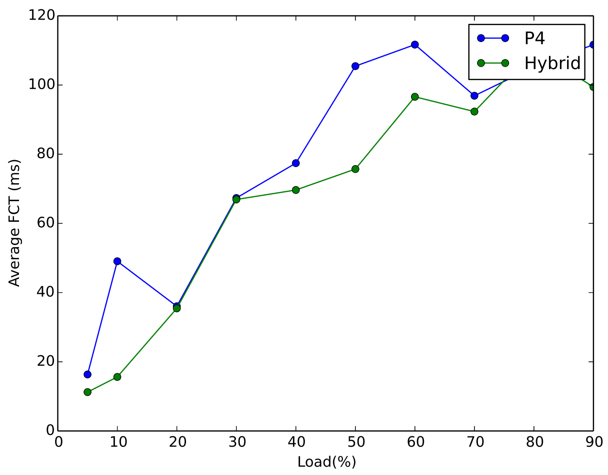

Figure 4: Average FCT

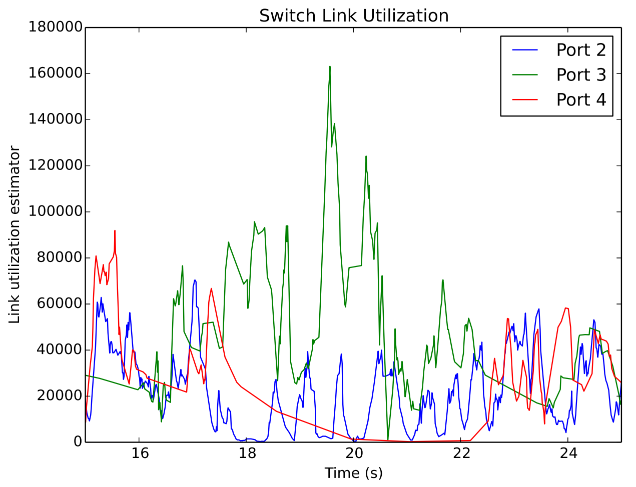

Figure 5: Link Utilization

In Figure 5, we observe the link utilizations of a TOR P4 switch under failure. The ports 3 and 4 are the upstream ports and when port 4 fails shortly after the instant 16s the traffic redirects to port 3.

6. RELATED WORK

CLOVE [8] is a load balancer implemented at the edge of the network. The system uses standard ECMP in the core and thus poses no requirements on the physical hardware in the network. It is an alternative to HULA rather than a mechanism for incrementally deploying it.

Incremental deployment of SDN for traffic engineering is explored in [7]. Their inspection of switch placement is relevant to our work. However, it operates on slower timescales because it goes through the controller. Our choice of programmable switch placement in the latency-sensitive scheme (§3.1) shares their spirit of placing special switches at the intersection of the most paths in the network. Panopticon [12] also explores the problem of partial SDN deployment, but targets enterprise policy enforcement rather than load balancing.

Past research related to network tomography [1, 5, 13] explored how we could infer fine-grained link statistics in the network core by performing end-to-end measurements. By applying network tomography techniques, one can estimate link delays, infer traffic source and destination or even identify network topology. We have not incorporated these ideas in our current design but instead leave it for future work.

7. FUTURE WORK

For future work, would be interesting to investigate possible applications of network tomography for our incremental deployment of programmable switches. In addition, would be of important value further investigation how can programmable switches improve the load balancing in generic networks that does not follow the nice structure of data centers networks.

In addition, more measurements should be performed on our proposed schemes. We did not analyze the performance of our Hybrid network under link failures. Moreover, for future work, would be interesting to explore how to combine the utilization information provided by programmable switches with link delay estimations.

As was proposed in [9], the programming interface provided by P4 is still not ideal and should be further investigated to facilitate the deployment of novel network applications. As example, in the present work, was necessary to modify the P4 reference switch to expose link utilizations to the network application. In addition, the current specification does not provide link information to allow the proper manipulation of queues or modification of the ECN flag what would allow easier interoperability with legacy switches. In other words, even though P4 provides a programmable interface, this interface should be robust enough such that most applications can be indeed implemented efficiently in the data plane without being subject to the cost of frequent hardware upgrades to achieve more functionality.

8. CONCLUSION

We proposed two schemes to the interoperability of programmable switches with legacy switches targeting load balancing on data center networks. These schemes would allow the incremental deployment of P4 switches in a network composed by legacy or OpenFlow switch without disrupting service. Our schemes explore end-to-end measurements of latency and data plane measurement of link utilization. Our results show that although we cannot achieve the same performance of networks fully composed of programmable switches the results are comparable. The work suggest further investigation of the performance and cost advantage of the incremental deployment of programmable networks.

9. REFERENCES

[1] A. Adams, T. Bu, T. Friedman, J. Horowitz, D. Towsley, R. Caceres, N. Duffield, F. L. Presti, S. B. Moon, and V. Paxson. The use of end-to-end multicast measurements for characterizing internal network behavior. *IEEE Communications magazine*, 38(5):152–159, 2000.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, Aug. 2008.

[3] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. Conga: Distributed congestion-aware load balancing for datacenters. *SIGCOMM Comput. Commun. Rev.*, 44(4):503–514, Aug. 2014.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[5] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: recent developments. *Statistical science*, pages 499–517, 2004.

[6] C. E. Hopps. Analysis of an equal-cost multi-path algorithm. 2000.

[7] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[8] N. Katta, M. Hira, A. Ghag, C. Kim, I. Keslassy, and J. Rexford. Clove: How i learned to stop worrying about the core and love the edge. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 155–161. ACM, 2016.

[9] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, SOSR '16, pages 10:1–10:12, New York, NY, USA, 2016. ACM.

[10] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[11] K. S. Lee, H. Wang, V. Shrivastav, and H. Weatherspoon. Globally synchronized time via datacenter networks. In *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM '16, pages 454–467, New York, NY, USA, 2016. ACM.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[13] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, et al. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 479–491. ACM, 2015.