Motivation for Adaptive and Dynamic Data Center Load Balancing

Lakshmi Krishnaswamy
University of California, Santa Cruz
Santa Cruz, CA, United States
lakrishn@ucsc.edu

Brevan Chun
University of California, Santa Cruz
Santa Cruz, CA, United States
bzchun@ucsc.edu

Nayan Bhatia
University of California, Santa Cruz
Santa Cruz, CA, United States
nbhatia3@ucsc.edu

Abstract

Data center networks are faced with traffic loads of varying characteristics. Load balancing algorithms are designed in order to assign flows to different paths ensuring high network utilization and congestion awareness. There have been a number of works that approach solving this problem in different ways. They are designed to be robust, sensitive to latency requirements,throughput guarantees, and resilient to faults. Recent works highlighting the growing trend of wide area network traffic competing within a data center alongside its data center network traffic have been gaining importance. However, this poses yet another unique challenge to data center network load balancer design. In this paper, we implement and analyze existing state of the art load balancers, and their performance on some synthetic and realistic workloads. We also evaluate their performance on wide area network traffic and evaluate their performance. Finally, based on our observations, we conclude on how the trends look like and on some promising future directions of work.

Keywords: Load Balancer, 5G, Data Center Network, WAN, TCP, Computer Networks, ECMP, Hermes

1 Introduction

With the large-scale deployments of 5G in the near future, there will be even more applications, including more bulk transfers of videos and photos, augmented reality applications and virtual reality applications which take advantage of 5G's low latency service. All these add to heavy, bulk of data being sent to the data centers and over the backbone network. These traffic have varying quality of service requirements, like low latency, high throughput and high definition video streaming. With the growth of cloud services that use data centers through wide area networks (WANs), there's an increase in the presence of WAN traffic along with data center (DC) traffic in data centers. These WAN flows are typically data heavy tasks that usually consist of backup data taken for a particular data center. The interaction of the data center and WAN traffic creates a very interesting scenario with its own challenges to be addressed. WAN and data center traffic are characterized by differences in the link utilizations, round trip times. Taking into account these contrasting characteristics of data center and WAN traffic and designing robust, network-aware load balancers is crucial in order to increase the performance of data center networks and increase network utilization. This work started with the vision of proposing a load balancer that is adaptive to the kind of traffic it encounters by learning from the network conditions and then predicting the optimal route for a given flow. Currently, we present the performance of couple load balancers on data center and WAN traffic respectively. We also present our discussions and conclusions and some future directions for this work.

1.1 Goals and Expectations

The big vision of this work was in serving as a motivation for redesigning load balancers, in order to adapt to different kinds of workloads and quality of service requirements in real time. However, in this paper, we present some preliminary results that focus on implementing and evaluating per-flow ECMP [7], per-packet ECMP and Hermes [9] on purely data center traffic and purely WAN traffic and expect to see them perform really well with data center traffic. In particular, we also expect Hermes to outperform both kinds of ECMP, as it is congestion aware, and does cautious rerouting. Even though Hermes makes use of real-time indicators to make decisions, because it was designed keeping data center traffic in mind, the expectation is that it would not work as optimally when faced with pure or even a combination of WAN traffic. This is to serve as a motivation to start thinking about redesigning load balancers and how they handle different kinds of traffic.

2 Related Works

Data centers deploy load balancing at different layers, pertaining to different levels of abstractions. In this work, our focus is mainly restricted to network level load balancers, which allocates flows across multiple links. Link level load balancers can be categorized based on the granularity of load balancing, congestion awareness and also based on their awareness of different flows. ECMP[7], Hermes[9], and Conga[5] are examples of network level load balancers.

Conga, a distributed load-balancing mechanism, implements its load-balancing in the network. This means that the switches in the network, report congestion and based on their feedback, Conga does a flowlet level switching. Flowlets are bursts of packets with related headers separated by a

time gap [5]. Conga is optimized for a two-tier topology, like the leaf-spine topology. It does benefit larger topologies however, it produces optimal results on a two-tier setup. This is a perfectly well designed and optimal scenario when there are only data center flows and covers most of the enterprise data center deployments. Conga is also shown to outperform ECMP under high loads as well as for both short and long flows. Also, Conga reacts to congestion at small round trip time scale, again well suited for data center flows only scenario.

These load balancers, and many others, do not differentiate between WAN and local traffic, but rather designed specifically for local traffic. A recent work, [8] highlights the growing trend of WAN traffic and its competence with data center traffic within a data center for resources. They highlight how WAN traffic and local datacenter traffic are characterized by different requirements. WAN traffic are characterized by longer round trip times, and thus in general high bandwidth delay product. Datacenter traffic changes at a much faster scale in comparison to WAN traffic. Thus, not accounting for these differences would lead to performance degradation and there could exist performance optimizations in doing this.

3 System Design

Simulation of a data center network was done using the OMNeT++ simulator [2] and INET network framework [1]. OMNeT++ is an event driven simulator program that can be utilized to simulate a multitude of systems. The INET framework enables network simulation for OMNeT++. We used OMNeT++ version 6.0pre11 and INET version 4.3.2 for all simulations.

The simulated network topology we used is the leaf-spine topology. Shown in figure 1, a leaf-spine topology consists of a layer of "spine" network switches and a layer of "leaf" switches. Each leaf switch is connected to each spine switch. Lastly, a set of servers is connected to a leaf switch. The servers generate workload requests. The different load balancers operate at the network level, and thus the load balancers are implemented in the TCP/IP stack. The load balancing algorithm runs on the network switches

3.1 ECMP

The "Equal Cost, Multiple Path" routing scheme[7] is a somewhat simple algorithm that manages the flow of packets through a network switch. A common version of the ECMP routing scheme uses hashing. ECMP with hashing works by using the header information of a packet to generate a hash key. The key then corresponds to a path in the hash table. This results in a per-flow routing scheme, since packets of the same flow will have the same packet header information (source and destination IP and port) and thus generate the same hash key and get routed to the same path. Packets with different header information are hashed differently and will be assigned a different path. We also implemented a per-packet version of ECMP[4]. Per-packet ECMP makes a new routing decision for every packet.

ECMP is an unaware routing scheme. Since it only utilizes static information, it does not make well informed routing decisions. Essentially, the routing of a flow is arbitrary. While this scheme then is able to evenly distribute flows to all of its paths, it is unable to react to dynamic load conditions in a network. Some flows will be more intensive than others, causing slowdowns on certain paths. ECMP is not able to compensate for these situations as the routing decision is based solely on static packet data.

The benefit of the ECMP routing scheme is mostly performance and simplicity. ECMP is a simple algorithm to implement and performs well in non-saturated networks. The routing decisions can also be made quickly, with low compute overhead and does not require special packet tagging or hardware. It is also worth noting that ECMP is one of the earlier network load balancer designs and is often used as the baseline for more advanced schemes [9][5].

3.1.1 Implementation.

The "Equal Cost, Multiple Path" routing scheme[7] can be implemented in many different ways - we chose to implement it with hashing. A hash key is generated from packet header data and the hash value is next hop path. This algorithm is deployed in every network switch. A path decision is only made for the next hop.

Per-packet ECMP is implemented by randomizing the equal cost paths of each packet[4]. Randomizing packets distributes the packets more evenly across the paths but will often result in packets needing to be reordered at the destination, because the paths have different latencies. Therefore, per-packet ECMP is not ideal for protocols that depend on packet ordering, like TCP. Implementing per-packet ECMP does not require hashing; packets are simply assigned a path randomly. Both of these algorithms are implemented at the network layer.

3.2 Hermes

Conversely, the Hermes routing scheme introduced in [9] is a context aware load balancer. Hermes is able to gather dynamic information about the network and make informed packet routing decisions - Hermes is a per packet routing algorithm. The original paper implementation uses round trip times (RTT) and explicit congestion notifications (ECN) to sense path congestion. RTT is simply the latency of a packet from one point to another. ECNs are attached to packets if a device is congested. For example, a network switch under load will have a finite sized queue that begins to fill up. Once the queue reaches a particular size, it marks the packets with ECNs. The ECNs can then be read when the marked packets are passed through other network devices. RTT and ECNs

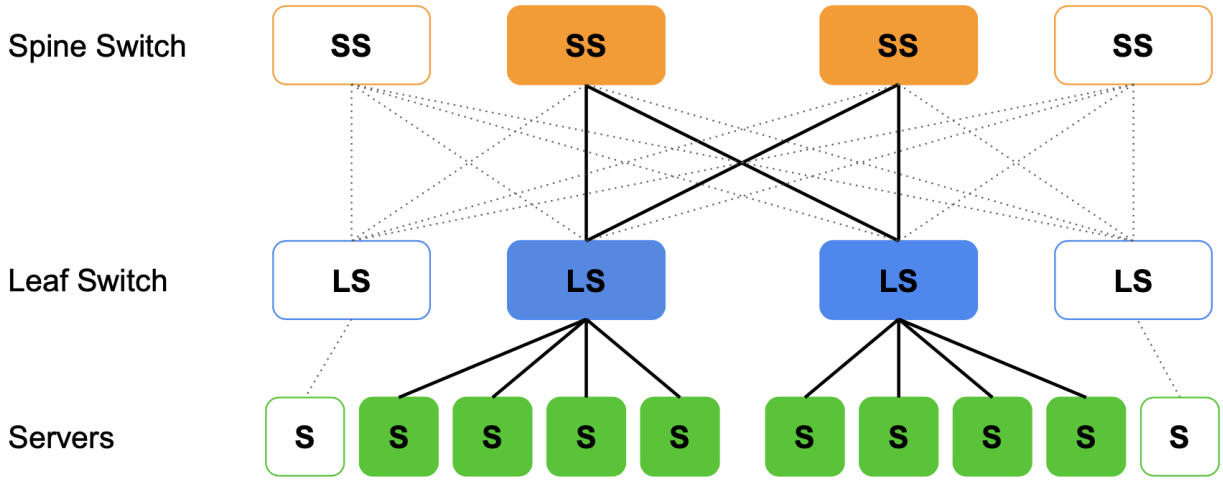

Figure 1. A 2x2 leaf spine topology (solid colors). A leaf spine network topology consists of a layer of spine switches, leaf switches, and servers. A number of servers are connected to a leaf switch and every leaf switch is connected to every spine switch.

are transport level signals and are implemented by adding tags to packet headers at network switches.

Hermes makes routing decisions based on both the RTT and ECN. If the RTT and ECN are below a certain threshold, the path is characterized as “good”. If the signals are both above a certain threshold, the path is characterized as “congested” . Any other case is characterized as a “gray” path (see table 1). Packets are routed on the best path possible and a new path is selected only when the current path becomes congested. In order to measure RTT and ECN, Hermes uses probing mechanism. The Hermes system generates packets to send across paths. Based on a set "probing interval", Hermes measures the RTT and checks for ECNs across paths, selected using the "power-of-two-choices technique" [9], at set intervals in time.

ECN	RTT	Path Characterization
High	High	Congested
~	~	Gray
Low	Low	Good

Table 1. Hermes path characterization

3.2.1 Implementation. Our Hermes implementation does not use probing and only uses RTT to make routing decisions. Paths are characterized as “good”, “congested”, and “gray” and packets are only routed to “good” paths. The RTT tags were implemented by adding tags to packets in the application layer. A real Hermes deployment adds packet tags at the transport layer, in the network switches. Adding packet tags in the application layer is unrealistic but was simpler to implement for use in simulations and achieves the same effect. In order to calculate the thresholds for path characterization, we make use of the baseline RTT of the network and heuristics recommended in the Hermes paper. For a ”good” path, since they have least congestion level, the end to end delay should be minimal and thus the threshold RTT is set to be around the baseline RTT. For a ”gray” path, we model the threshold to be about 1.5times the baseline RTT, in order to model a fairly congested path. For ”congested” paths, the threshold is set to be at least two times the baseline RTT. Since these values are set heuristically, there is a lot of scope for fine tuning and optimizations.

4 Evaluations

We use a set of workloads for running our simulations. First, we have the bi-modal distribution workload - which is a synthetic workload trace. It consists of short flows that 10KB in size and long flows that are 1MB in size. In order to mimic the heavy-tailed distribution of data center workloads, we have 750 flows of 1MB and 49250 flows of 10KB, with the average flow size of about 19.901KB.

For realistic workloads, we use two different kinds of workloads: web-search workload [3] and data-mining workload [6]. The web-search workload consists of mostly latency-sensitive tasks. Many of the flows are short and latency sensitive flows, with some large and bandwidth sensitive flows which are usually greater than 1MB. One of the common challenges with such workloads are the large flows blocking “bursty” small flows. The data-mining workload has a more, skewed and heavy-tailed distribution. 99% of flows are less than 100MB while, 90% of bytes are in flows greater than

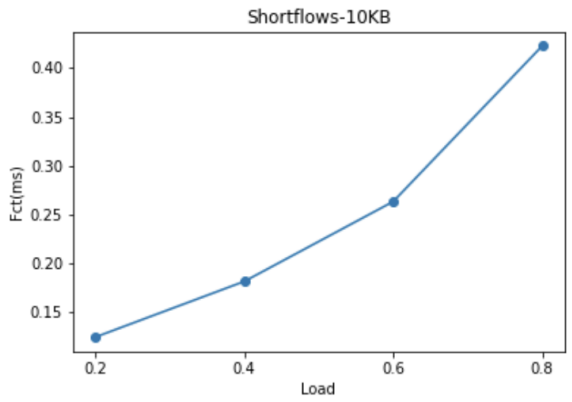

Figure 2. Effect of per-flow ECMP on short flows

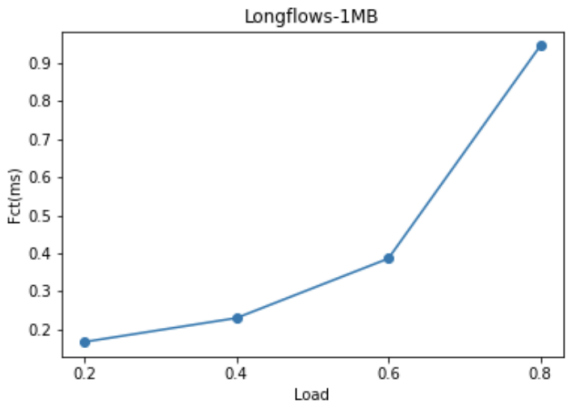

Figure 3. Effect of per-flow ECMP on long flows

100MB. Thus, the bulk of the flow size comes from just 1% of the flows. In order to simulate for WAN traffic, it would definitely be interesting and more realistic to use traces of more recent applications like machine learning workloads, video streaming workloads. However, in this paper we have focused on first benchmarking WAN traffic purely using link parameters, as described in [8].

4.1 Results and Discussions

For our evaluations, we conducted three different sets of experiments. In experiment 1, we wanted to benchmark the performance of the load balancers on some synthetic workloads first, in order to verify its implementation correctness. In experiment 2, load balancers are evaluated on data center traffic only. In experiment 3, load balancers are evaluated on WAN traffic only.

4.2 Experiment 1

In this experiment, we test the implementation of per-flow ECMP. we use an 8x8 leaf-spine topology network, that has 128 servers. Each link in the network has a delay of 10μs, and therefore the network has a baseline RTT of about 80μs. The workload used here are the synthetic bi-modal workload described above. Network load was characterized by having the same number of 50,000 flows but varying their arrival rates. Higher loads translate to having the flows arrive in close proximity to each other. With an increase in load in the network, the time taken to complete a flow (defined as flow completion time) should increase.

From figure 2 and 3, we see that the flow completion time increases with network load. Since the baseline RTT is about 80μs, accounting for transmission and propagation delays, the theoretical expected flow completion time for a 10KB flow is about 90us. We see from fig 2 that when the network is lightly loaded, the flow completion time is about 75μs, which is fairly close to expected value. For long flows, the observed flow completion time is much lower than the expected baseline of about 600μs, which needs to be investigated further. Calculations for our expectations can be found in appendix A.1.

4.3 Experiment 2

In this experiment, we test all three load balancers, on purely DC traffic. In order to account for intra-DC setting, we set all the links to have a delay of 10μs. We use the data-mining workload described above. The leaf-spine topology was used, with 8 servers in total. Network load was characterized by having the same number of 500 flows but varying their arrival rates. Higher loads translate to having the flows arrive in close proximity to each other.

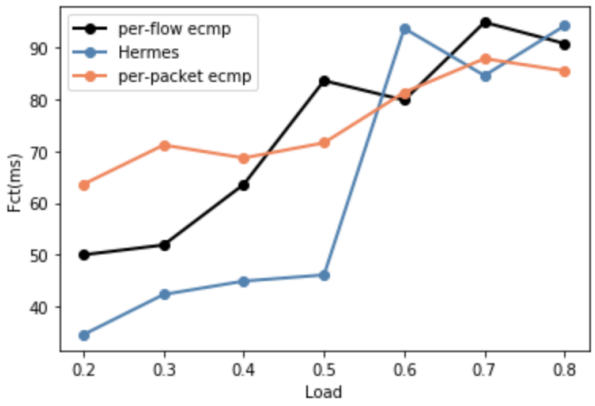

Figure 4. A direct comparison of per-flow, per-packet ECMP, and Hermes routing schemes with local data center traffic. Hermes performs better than ECMP until 60% load, where all load balancers perform similarly.

We expected Hermes to perform better than both variations of ECMP, which is indeed the observation as well. This is promising, given that only a rudimentary form of Hermes was implemented, with just an indication of end to end delay.

The fluctuations in flow completion times with increase in load, is also due to randomness in the simulations that could be smoothed out on repeating multiple runs and taking an average of them. Due to time constraints, this is part of our immediate future work.

4.4 Experiment 3

In this experiment, we test all three load balancers, on purely WAN traffic. In order to account for WAN setting, we set all the links to have a delay of 10ms. We use the data-mining workload described above. We do want to explore machine learning workloads, real-time video streaming workloads as part of our future work. The leaf-spine topology was used, with 8 servers in total. Network load was characterized by having the same number of 500 flows but varying their arrival rates. Higher loads translate to having the flows arrive in close proximity to each other.

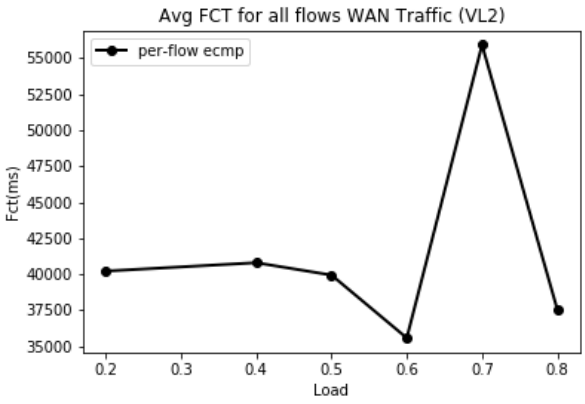

Figure 5. Performance of per-flow ECMP with WAN traffic

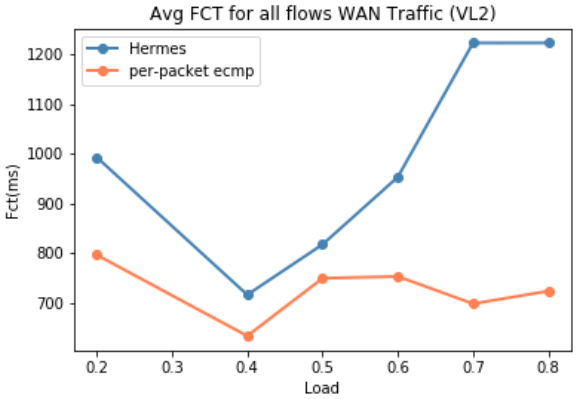

Figure 6. Performance of per-packet ECMP and Hermes with WAN traffic

It can be observed that per-flow ECMP performs really bad, with flow completion times in the magnitude of about 35000-60000ms. It also exhibits inconsistencies with increase in network load. Since per-flow ECMP hashes at the flow level, given the high RTT of WAN traffic, this is a very high granularity of load balancing and thus results in lot of flow collisions, even when the network is lightly loaded. It was surprising to observe that in case of WAN traffic, per-packet ECMP seems to perform consistently better than Hermes and per-flow ECMP. The high flow completion times for 20% network load, especially for Hermes, could indicate that because it identifies a "good" path, it continues sending all packets through the same path. This highlights the need for implementing an active probing mechanism even for lightly loaded networks.

5 Conclusion

With normal, local DC traffic, a rudimentary implementation of Hermes performs better than both versions of ECMP. We expected this result but Hermes has richer features that need to be explored further, and highlights the need for finding ways of tuning for the best parameters for a given traffic type. This result also shows that any high performance load balancer should take advantage of dynamic network information to make informed routing decisions.

When introduced to WAN traffic, all load balancers performed significantly worse. Our simulation results show us that Hermes performs worse than per-packet ECMP - an unexpected result. Potential reasons for this are discussed in the analysis but it shows that WAN traffic produces unpredictable performance from these load balancers that were originally designed for local data center traffic. One of the future goals of this work includes the design of a load balancer than can perform well with both local and WAN traffic. It is clear from our simulations that there is a need for this.

6 Lessons Learnt

One of the most important lessons learned in this research was that the simulator environment was flawed. For example, numerous simulation runs did not complete the desired number of flows. Having not completed as many flows as intended, the flow completion time would be much lower than expected - appearing as unexpectedly high levels of performance. We think this may have been the result of packet/flow drops because of excessive latency and load in the network. We were not able to fully confirm this hypothesis before the writing of this report. We tried to simulate for same number of flows (500), but with a bigger topology of having 32 servers, but these still did not give us optimal performance. This also underlines the importance of designing correct ways of modelling congestion levels in the network.

Additionally, large scale simulations were time consuming and prevented us from efficiently debugging tests. This calls for the need for more powerful compute resources. We tried experimenting with fork and virtual machine to speed up the

simulations. This also made us realize about the importance and need for designing a good set of microbenchmarks; ones that are quick to run while also being somewhat realistic. This would allow us to debug the simulation more efficiently while also providing us with useful data.

7 Future Work

This paper evaluates current load balancers' performance on data center and WAN traffic. The immediate next steps include evaluating their performance using combination of WAN and data center traffic, and observing for any performance changes. We are also looking at modelling WAN traffic differently, as currently they are only characterized by different link parameters. One way (as Prof.Quinn highlighted), would be to use some actual traces that are sent over WAN. Once we have these implemented and evaluated, we would like to design an straw-man approach of evaluating the impact of different parameters (like hash value for ECMP, RTT and ECN threshold, sending rate and probing intervals with Hermes, etc) with data center and WAN traffic. We then want to use this as our input to propose a new design of load balancers that can adapt to the best parameters needed when faced with a given kind of traffic. We would also like to look at ways of labelling and eventually learning what kind of traffic is the load balancer currently dealing with.

8 Acknowledgement

We would like to thank professor Andrew Quinn for his continued help and support throughout the quarter and guiding us with helpful feedback and directions.

References

[1] [n.d.]. INET Framework. https://inet.omnetpp.org.

[2] [n.d.]. OMNeT++ Discrete Event Simulator. https://omnetpp.org.

[3] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al. 2010. Hedera: dynamic flow scheduling for data center networks.. In *Nsdi*, Vol. 10. San Jose, USA, 89–92.

[4] Mohammed Alasmar and George Parisis. 2019. Evaluating modern data centre transport protocols in OMNeT++/INET. *Proceedings of 6th International OM* 66 (2019), 1–10.

[5] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*. 503–514.

[6] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. 51–62.

[7] Christian Hopps et al. 2000. *Analysis of an equal-cost multi-path algorithm.* Technical Report. RFC 2992, November.

[8] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, and Amin Vahdat. 2020. Annulus: A Dual Congestion Control Loop for Datacenter and WAN Traffic Aggregates. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 735–749. https://doi.org/10.1145/3387514.3405899

[9] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. 2017. Resilient datacenter load balancing in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 253–266.

A Appendix

A.1 Experiment1 - theoretical calculations

We have all links to be of 10Gbps.
Packet size is 1500 bytes.
Initial congestion window size is set to 15KB.
Each link has 10us link delay.
The transmission of one packet : $(1500 \cdot 8)/(10 \cdot 10^9) = 1.2us$
Any flow will have $(10 + 1.2) \cdot 8 = 89.6\mu$s delay
Flows that have source and destination on same switch will have $11.2 \cdot 4 = 44.8\mu$s