Deterministic Bandwidth-Based Packet-Level Traffic Splitting for Datacenter Networks

Cheng-Yu Wu, Li-Hsing Yen, Ping-Chun Hsieh, and Chien-Chao Tseng
Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan.
Email: {*cywu.cs06g, lhyen, pinghsieh*}*@nctu.edu.tw*, cctseng@cs.nctu.edu.tw

***Abstract*—Traffic splitting is to distribute traffic over multiple paths so as to better utilize link bandwidth and avoid potential link congestion. Many traffic splitting schemes have been proposed for datacenter networks, which provide different levels of splitting granularity. This paper proposes a packet-level traffic splitting scheme, which provides the most fine-grained link utilization as it lays out a forwarding path for each individual packet. The proposed scheme mitigates possible packet reordering problem by equalizing the build-ups of packet queues in all switches as the path layout utilizes links in proportional to the associated exploitable link bandwidth. The result is a packet dispatching rule designated to each switch. We implemented the rules using P4 (Programming Protocol-Independent Packet Processors) switches and conducted experiments to measure the performance. Experimental results show that the proposed scheme provides higher goodput compared with another packet-level traffic splitting scheme which dispatches packets randomly among all links.**

I. Introduction

Bandwidth demands in datacenter networks have been steady increasing [1] due to the introduction of new applications such as interactive web services, online streaming, video on demand, and large amount of sensory data generated by Internet-of-Thing (IoT) devices. A technical strategy to keep up with increasing bandwidth demands is to efficiently manage flow of traffic across the datacenter network. An effective traffic control scheme can improve bandwidth utilization and thus reduce bandwidth provisioning costs and increase revenues. Moreover, high bandwidth utilization also enhances user experience by providing high throughput and short response time to users.

Many traffic control techniques, such as transmission control, rate limiting, packet pacing, load balancing, multipathing, and scheduling, have been proposed for bandwidth utilization enhancement [2]. This study considers a multipathing-based technique—*traffic splitting*. Traffic splitting is to distribute the traffic between a pair of source and destination hosts over multiple paths so as to better utilize link bandwidth and avoid potential link congestion. Many traffic splitting schemes have been proposed for datacenter networks. Different approaches provide different levels of splitting granularity. Theoretically speaking, a fine-grained traffic splitting could provide a high level of load balancing. On the other hand, traffic splitting creates multiple transmission paths between a pair of hosts, which may cause *packet reordering* [3] problem. In particular, packet reordering may significantly degrade the performance of TCP connection due to the triggering of TCP congestion control by fast retransmit [4]. Several studies [5], [6] have discussed the issues caused by packet reordering and evaluated the impact of packet reordering on performance.

Traffic splitting can be performed on flow, flowlet, or packet level. Flow-level traffic splitting splits traffic by flows, which provides the coarsest splitting granularity but avoids the packet reordering problem. Flowlet-level traffic splitting provides a finer splitting granularity than the flow-level traffic splitting. A flowlet is a burst of packets within a transport-layer flow [7]. Two consecutive flowlets in the same flow are separated by a time gap greater than a threshold value δ. Therefore, as long as δ is larger than the maximum latency of the paths between the source and the destination, packet reordering across flowlets are impossible even if different flowlets go through different paths. However, there are still some issues associated with flowlet-level traffic splitting. First, flowlet-level traffic splitting can hardly split bursty traffic into flowlets [8]. Second, the maximum latency of the paths between the source and the destination may vary dynamically. If δ does not adapt to network dynamics, packet reordering may still occur. Packet-level traffic splitting scheme selects a forwarding path for each individual packet, which provides the finest splitting granularity but does not guarantee orderly packet arrivals.

This paper proposes a new packet-level traffic splitting scheme, named deterministic bandwidth-based (DBB), which reserves paths for packets based on available link bandwidth. The goal is to exploit each available link with an intensity proportional to the available link bandwidth. Doing so can hopefully equalize the build-ups of packet queues in all switches in the datacenter network and thus mitigates the possibility of packet reordering. This strategy is effective particularly in Clos-based networks [9] such as fat-tree, VL2, and leaf-spine networks, where multiple paths of equal length can be found for a particular source-sink switch pair because the topology is symmetric and multi-staged.

As the name suggests, DBB differs from existing packet-level splitting schemes such as Random Packet Spraying (RPS) [10] in that it elaborates (rather than randomly determines) a forwarding path for each packet in a cyclic manner so that we could evenly interleave all packets in a dispatching cycle and thus prevent packets from accumulated on a particular path. We implemented DBB on P4 (Programming Protocol-Independent Packet Processors) switches with two implementation options and conducted experiments to measure the performance of DBB. The results indicate that, compared with RPS, the proposed approach achieves higher total goodput in all the experimental settings. The results confirm the ability of DBB to mitigate potential packet

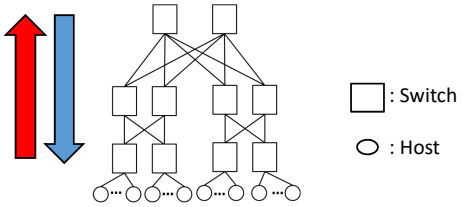


Fig. 1: A simple VL2 network topology

reordering problem.

The remainder of this paper is structured as follows. Sec. II briefs the backgrounds and related work. Sec. III presents the proposed scheme from the concept to the details of the design. Sec. IV presents the experimental results with P4 switches and the last section concludes this paper.

II. Background and Related Work

Datacenter networks are usually multi-staged with tree-structured topologies. In this study, we assume a multi-staged network with identical out-degree (i.e., the number of egress links) in all switches in the same stage as exemplified in Fig. 1.

Flow-level traffic splitting identifies a unique transport-layer flows by five tuples (source IP address, destination IP address, source port, destination port, and protocol) in the packet header. It provides the coarsest splitting granularity as all packets of a transport-layer flow go through the same path. A merit of it is that no packet reordering will occur for the reason of different packet transmission paths.

Equal-cost multi-path (ECMP) forwarding is the most common flow-level traffic splitting scheme used in datacenter networks. ECMP treats every flow equally and does not consider link congestion status. Therefore, it is likely to cause unbalanced link utilization when both elephant and mouse flows coexist. Furthermore, multiple flows may map to the same path through the hash function used by ECMP schemes, leading to unbalanced link utilization. Hopps [11] analyzed the performance and the disruption caused by the hashing-based algorithm of ECMP. Cao et al. [12] analyzed the performance of different hash-based schemes for load balancing.

Flowlet-level traffic has a greater potential to achieve load balancing if the assignment of paths to flowlets is congestion-aware. Congestion-aware designs, on the other hand, induce additional efforts in collecting and processing network status and feedbacks of congestion information. The overhead may make these schemes not fast enough to react to network dynamics. Distributed Congestion-Aware Load Balancing for Datacenters (CONGA) is a flowlet-level congestion-aware load balancing scheme that collects congestion metrics on each path based on leaf-to-leaf feedbacks. CONGA approaches load balancing by letting a leaf switch assign a path with the minimum link utilization to a new flowlet. A downside of CONGA is the demand of considerable storage space from each leaf switch to keep the congestion information of every path. Moreover, per-path congestion information based on the leaf-to-leaf feedback may not be timely due to the round-trip time needed for such a feedback. CONGA may also overwhelm a downstream switch when multiple paths from different leaf switches join at that switch.

Hop-by-Hop Utilization-Aware Load Balancing Architecture (HULA) is another flowlet-level congestion-aware load balancing scheme that uses probes to collect congestion information. Unlike CONGA, which keeps congestion information of all paths on every leaf switch, HULA maintains congestion information only for the best next-hop switch on all switches. However, HULA needs additional probes to gather link utilization and update the best next hop on each switch. The frequency of probing is a trade-off in HULA. A low probing frequency cannot detect congestion in time whereas a high probing frequency may consume considerable link bandwidth.

Many researchers have proposed packet-level traffic splitting schemes [10], [13], [14] to improve the level of traffic balancing. Random packet spraying (RPS) spreads all packets of a flow uniformly at random among different shortest paths [10]. The experiment results in [10] showed that flow-based ECMP and RPS have similar throughput for small flows, while RPS tends to yield higher throughput than ECMP for large flows. It was reported that RPS achieves shorter average flow completion time (FCT) for short flows and higher average throughput for long flows over the single path used by TCP [13]. However, [14] reported that due to the trait of randomness, RPS would not distribute packets evenly over all paths. Therefore, queues would still build-up, and the number of out-of-order packets would increase, which causes poor performance for TCP connections.

III. Proposed Scheme

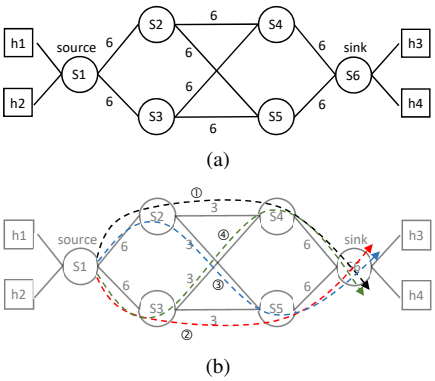


Fig. 2: An example illustrating the idea of the proposed approach. (a) the input topology (b) the output result

The basic idea of the proposed approach is to let each switch cyclically dispatch all ingress packets belonging to a particular source-destination host pair over all its egress links following a predetermined rule so that different packets go through different paths in a way that all links involved in these paths are proportionally utilized to achieve load balance. Fig. 2a shows an example of layer-2 topology in a datacenter network with S1 and S6 being the source and sink switches, respectively. The number beside each link indicates the link capacity. Fig. 2b shows the result of the proposed approach, where four possible paths have been created for all the packets from S1 to S6. The number beside

each link indicates the exploitable bandwidth associated with the link that maximizes the network flow from S1 to S6. A distinct characteristic of the result is that the number of packets passing through a particular link is proportional to the exploitable link bandwidth if the first packet follows path 1, the second packet follows path 2, and so on. This characteristic achieves load balancing via multipathing while maximizing traffic flow from S1 to S6.

The main challenge of our task is to find out and set up corresponding packet dispatching rules for all switches involved in the solution. To ease the task, we sequentially number every packet from the same source switch to the same sink switch. Our goal is then to determine a *packet dispatching cycle* (PDC) with length N such that each switch processes the i-th packet as the $(i+N)$-th packet for all positive integer i, $1 \leq i \leq N$. To this end, DBB acquires the network topology with link bandwidth information, divides the network into stages, and finds out the exploitable bandwidth of each link by solving the maximum network flow problem. With that information, DBB can then determine the packet dispatching rules for all switches in a stage-by-stage manner such that the number of packets dispatched to any link in a PDC is proportional to the exploitable bandwidth associated with it. All the rules are then converted to switching rules and deployed into the corresponding switches.

The following are details of the proposed scheme. We use the topology shown in Fig. 2a as an example in the following.

A. Finding Exploitable Link Bandwidth

To maximize bandwidth utilization, DBB first finds out the maximum network flow from the source to the sink switches. This can be done by linear programming (or linear optimization). The solution determines the set of all switches involved and the exploitable bandwidth associated with each link.

For example, the maximum network flow from S1 to S6 in Fig. 2a is $z = 6$. For this solution, Fig. 2b shows the exploitable link bandwidth associated with each link $(\mathrm{S}i, \mathrm{S}j)$, $b_{i,j}$ ($b_{1,2} = b_{1,3} = b_{4,6} = b_{5,6} = 6$ and $b_{2,4} = b_{2,5} = b_{3,4} = b_{3,5} = 3$.)

B. Stage-by-Stage Packet Dispatching

Even we know the exploitable bandwidth of each link, it is still difficult to find out the set of all paths that collectively utilize every link in proportional to the associated exploitable link bandwidth. Fortunately, we can divide the whole network into stages, each of which has identical aggregated bandwidth, and arrange the packet dispatching rule in a stage-by-stage manner.

Let S be the set of all switches that are included in the solution to the maximum flow problem. The first step is to partition all switches in S (together with their egress links) into stages. Let l denote the total number of stages. Stage 1 includes only the source switch (together with all of its egress links). Stage 2 includes all switches in S (together with their egress links) that are one-hop away from the source switch. Similarly, stage 3 includes all switches in S that are one-hop away from a Stage-2 switch, and so on. Fig. 3 shows the stage partition for the network topology shown in Fig. 2a.

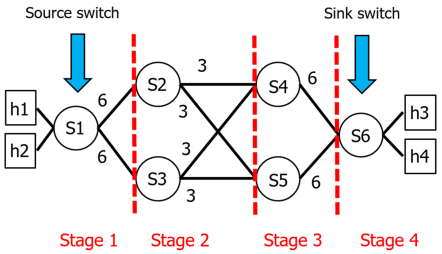


Fig. 3: Dividing the network topology into multiple stages

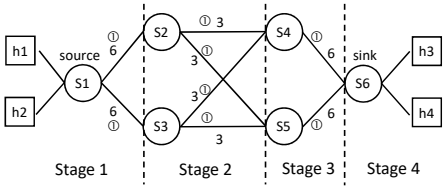


Fig. 4: SIR of each link (circled number)

The next step is to derive a minimum number of packets that comprise a PDC in each stage. It involves calculating the ratio of the exploitable bandwidth on each egress link in a stage to the aggregated exploitable bandwidth on all egress links in the same stage. Let $L(s)$ denote the set of all egress links in stage s. For each link $(i, j) \in L(s)$, that ratio is $r_{i,j} = b_{i,j} / \sum_{(u,v)\in L(s)} b_{u,v}$. We multiply each ratio $r_{i,j}$ by a minimum constant so that the ratios in the same stage are all integers. We call the result a simple integer ratio (SIR) and use $\rho_{i,j}$ to represent the result of link (i, j). For example, the SIR between $b_{1,2}$ and $b_{1,3}$ in our example is $1 : 1$ so $\rho_{1,2} = \rho_{1,3} = 1$. Fig. 4 shows $\rho_{i,j}$ for each link (i, j).

We want to dispatch all packets coming into stage s to all egress links in stage s in a manner such that the bandwidth consumed on each link $(i, j) \in L(s)$ is in proportional to $b_{i,j}$. The idea is to dispatch exactly $\rho_{i,j}$ packets to link (i, j) in a PDC. This implies that the PDC in stage s comprises $\rho^s = \sum_{(u,v)\in L(s)} \rho_{i,j}$ packets. In our example, the PDC in stage 2 comprises 4 packets.

C. Determining Network-Wide Packet Dispatching Cycle (PDC)

Because the value of ρ^s is different in different stage s, the minimal number of packets comprising the PDC for the whole network is the least common multiple (LCM) of all ρ^s's. Formally,

$$N = \mathrm{lcm}(\rho^1, \rho^2, \cdots, \rho^{l-1}). \tag{1}$$

In our example, $N = \mathrm{lcm}(2, 4, 2) = 4$.

Let $N_{i,j}$ be the number of packets to be dispatched through link $(i, j) \in L(s)$ in a network-wide PDC. We have

$$N_{i,j} = \rho_{i,j} \frac{N}{\rho^s}. \tag{2}$$

In our example, $N_{1,2} = 1 \times 4/2 = 2$ while $N_{2,4} = 1 \times 4/4 = 1$. Therefore, two and one packets should be dispatched from S1 to S2 and from S2 to S4, respectively, in a network-wide PDC.

TABLE I: Packet Dispatching for S1

Packet Seq.	S2		S3		Next-Hop Switch
	$Q_2(2)$	$R_2(2)$	$Q_2(3)$	$R_2(3)$	
1	2	2	2	2	S2
2	2	1	2	2	S3
3	2	1	2	1	S2
4	2	0	2	1	S3

TABLE II: Packet Dispatching for Stage-2 Switches

Packet Seq.	Current Switch	S4		S5		Next-Hop Switch
		$Q_3(4)$	$R_3(4)$	$Q_3(5)$	$R_3(5)$	
1	S2	2	2	2	2	S4
2	S3	2	1	2	2	S5
3	S2	2	1	2	1	S5
4	S3	2	0	2	1	S4

D. Packet Dispatching Rule for Each Switch

To lay out the packet dispatching rule for each switch, we need to find out the *quota* of each switch, i.e., the total number of packets to be dispatched to the switch in a network-wide PDC. Formally, the quota of a switch k in stage s, $2 \le s \le l$, is

$$Q_s(k) = \sum_{(i,k)\in L(s-1)} N_{i,k}. \tag{3}$$

We also need to know how many packets are yet to be dispatched to each switch k. Denote it by $R_s(k)$, which is $Q_s(k)$ minus the number of packets that have already been scheduled dispatching to switch k. When a switch in stage s needs to pick up a next-hop switch in stage $s+1$ for packet dispatching, it simply selects a switch j that has the highest $R_{s+1}(j)$ to $Q_{s+1}(j)$ ratio (and break ties arbitrarily).

Algorithm 1 Stage-by-Stage Packet Dispatching

Require: Network Topology
Ensure: Deterministic Path Table **p**
1: $l \leftarrow$ number of stages in the topology
2: Calculate SIR $\rho_{i,j}$ for each link (i, j)
3: $\rho^s \leftarrow \sum_{(u,v)\in L(s)} \rho_{i,j}$ for all $1 \le s \le l-1$
4: $N \leftarrow \mathrm{lcm}(\rho^1, \rho^2, \cdots, \rho^{l-1})$
5: $N_{i,j} \leftarrow \rho_{i,j} N/\rho^s$ for each $(i, j) \in L(s)$ and $1 \le s \le l-1$.
6: $Q_s(k) \leftarrow \sum_{(i,k)\in L(s-1)} N_{i,k}$ for all k and $2 \le s \le l$
7: $R_s(j) \leftarrow Q_s(j)$ for all k and $2 \le s \le l$
8: **for** $seq = 1$ to N do **do**
9: **for** $s = 1$ to $l-1$ **do**
10: $k \leftarrow \arg\max_j \{R_{s+1}(j)/Q_{s+1}(j)\}$
11: $\mathbf{p}[seq, s] \leftarrow k$
12: $R_s(k+1) \leftarrow R_s(k+1) - 1$
13: **end for**
14: **end for**
15: **return p**

Algorithm 1 details how to complete a PDC. Each forwarding path is actually constructed in a link-by-link manner. The source switch S1 in our example needs to select either S2 or S3 as the next-hop switch for all outgoing packets. As shown in Table I, the algorithm picks up S2 for the first packet ($seq = 1$) and S3 for the second one ($seq = 2$). The reversed order is also possible because S2 and S3 have the same priority initially.

Suppose that the first packet goes to S2, which needs to select either S4 or S5 as the next-hop switch. Here S4 and S5 also have the same priority initially. The result shown in Table II assumes the selection of S4. S4 has only one selection, S6, for the next-hop switch. This completes the path for the first packet. Table III shows the paths found for all four packets in a network-wide PDC.

Figure 5 shows another topology with uneven link bandwidth. Here $N = \mathrm{lcm}(5, 10, 5) = 10$, so there are 10 packets in a PDC. The result is shown in Table IV.

TABLE III: Complete PDC for the Topology Shown in Fig. 2a

Packet Seq.	Stage 1	Stage 2	Stage 3	Stage 4
1	S1	S2	S4	S6
2	S1	S3	S5	S6
3	S1	S2	S5	S6
4	S1	S3	S4	S6

E. Conversion to Switch Rules

Since the route of a packet depends on the sequence number of the packet, every switch needs a packet sequence counter for packet dispatching. There are two design options. A switch may uses a *shared counter* for all packets of the same source-sink switch pair. Alternatively, a switch may keep an *independent counter* for each distinct flow between the same pair. The latter design actually conducts packet-level traffic splitting for each flow. We will evaluate the performance of these two different designs numerically in the next section.

IV. Numerical Results

We implemented DBB using P4 switches (Tofino ASIC [15]). P4 switches allow for customized headers, parsers, and match-action tables, which facilitate the implementation of the proposed approach. For example, we used the registers of P4 switches to implement circular counters to identify the ordinal numbers of ingress packets. We implemented DBB with both shared and independent counters and conducted experiments for performance comparisons with RPS [10].

We used 8 Barefoot Tofino Switches (INV D10056) and 3 QCT D51B-1U servers as traffic generators. Fig. 6 shows the topology. The link capacity is uniformly 10 Gbps. We considered different scenarios by varying the traffic loads, the numbers of connections, the rates of connections, and the packet sizes of connections. We used iPerf3 to establish TCP connections between end hosts, generate traffic according to the designated target rates, and measure the sending rates (i.e., goodput) at the other ends of the connections. We used TCP connections instead of UDP datagrams because TCP connections react to packet reordering so the measured goodput could reveal whether the proposed approach is impacted by the packet reordering problem.

TABLE IV: Complete Dispatching Cycle for the Topology Shown in Fig. 5

Packet Seq.	Stage 1	Stage 2	Stage 3	Stage 4
1	S1	S2	S4	S6
2	S1	S3	S5	S6
3	S1	S3	S5	S6
4	S1	S3	S4	S6
5	S1	S3	S5	S6
6	S1	S2	S5	S6
7	S1	S3	S4	S6
8	S1	S3	S5	S6
9	S1	S3	S4	S6
10	S1	S3	S5	S6

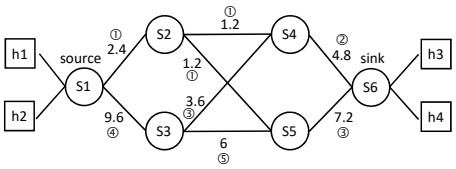


Fig. 5: Topology with uneven link bandwidth. Circled number indicates the SIR of the associated link.

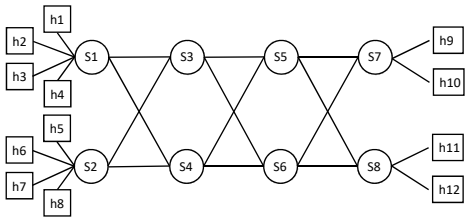


Fig. 6: Topology for experiments with Barefoot Tofino Switches

A. *High and Low Traffic Loads*

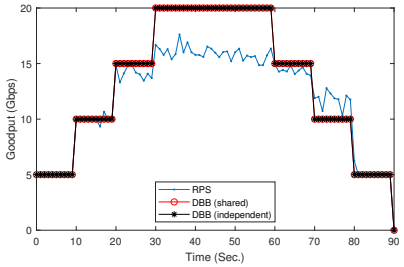
The first experiment established one TCP connection for each of the four source-destination host pairs: (h1, h9), (h3, h11), (h6, h10), and (h8, h12). We tested two TCP target rates: 5 Gbps and 10 Gbps. The aggregated traffic loads with these two rates are thus 20 Gbps and 40 Gbps, respectively. With 10-Gbps link capacity, the bandwidth capacity of each stage is 40 Gbps. Therefore, these two settings generated traffic loads equivalent to 50% and 100% bandwidth capacities, respectively.

We created a new connection every 10 seconds, and each connection lasted for 60 seconds. Fig. 7 shows the sending rate collected by iPerf3 with both settings of load. For the proposed scheme (DBB), we can see the total sending rate increased when a new connection was added into the network. When all connections coexisted (during the interval from the 30th to the 60th seconds), the total sending rates of DBB were higher than RPS. The average goodputs of RPS, DBB (shared counter), and DBB (independent counter) during this interval were 15.94, 20.00, and 20.00 Gbps, respectively, with 20 Gbps traffic load and 15.77, 36.63, and 37.03 Gbps, respectively, with 40 Gbps traffic load. We can see that the sending rate of RPS did not even reach the half of the network capacity (i.e., 20 Gbps).

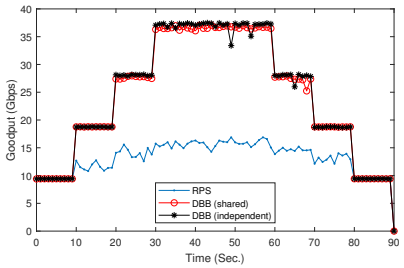
B. *Number of Connections*

We had two settings for the number of connections. In the first setting, we created 10 connections between each pair of source and destination hosts, resulting in 40 connections in total. The target rate of each connection was set to 1 Gbps. In the second setting, the number of connections was increased to 100 connections per pair (resulting in 400 connections in total) with target rate 0.1 Gbps per connection. All connections commenced at the very beginning and lasted for 90 seconds.

Figure 8 shows the sending rates collected by iPerf3 in these two settings. The average sending rates of RPS, DBB (shared counter), and DBB (independent counter) were 18.87, 37.48, 37.57 Gbps, respectively, with 40 connections and

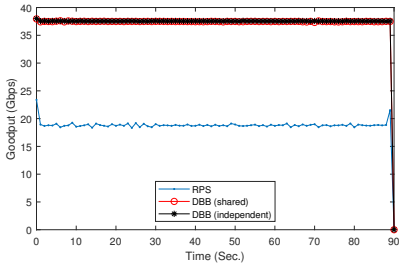


(a)

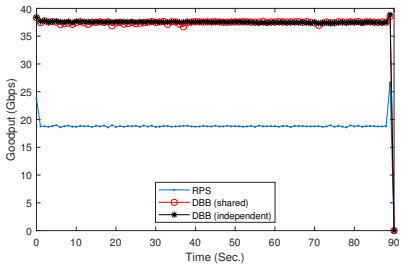


(b)

Fig. 7: Goodput with different traffic loads. (a) 20 Gbps (b) 40 Gbps



(a)



(b)

Fig. 8: Goodput with two different numbers of connections. (a) 40 connections (b) 400 connections

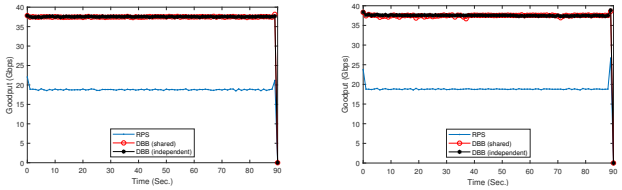
18.93, 37.50, 37.55 Gbps, respectively, with 400 connections. We observed no significant performance difference between these two settings because these two settings injected the same total amount of traffic load and all the tested schemes perform packet-level traffic splitting. Nevertheless, the proposed scheme (DBB) significantly outperformed RPS in both settings.

C. *Heterogeneous Connection Rates*

We created 40 and 400 connections and set up the target rate of each connection by the following rule. The basic rates of a connection were 2 Gbps and 0.2 Gbps in the 40-connection and 400-connection settings, respectively. We used a ratio vector *bw_ratio* with contents

$[0.5, 0.1, 0.9, 0.2, 0.8, 0.3, 0.7, 0.4, 0.6, 0.5]$ to set up the target rate of each connection. The target rate of the i-th connection was set to the basic rate times $bw\_ratio[i \mod 10]$ for all i.

Figure 9 shows the sending rates collected by iPerf3 with heterogeneous connection rates. The average goodputs of RPS, DBB (shared counter), and DBB (independent counter) were 18.84, 37.50, 37.54 Gbps, respectively with 40 connections and 19.07, 37.62, 37.67 Gbps, respectively with 400 connections. The results are similar to those of the previous experiment because the total traffic loads remain the same. Here the proposed scheme (DBB) still outperform RPS.



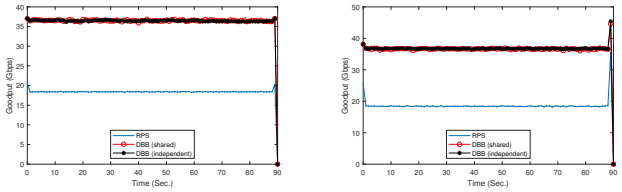
(a) (b)

Fig. 9: Goodput with heterogeneous connection rates. (a) 40 connections (b) 400 connections

D. Heterogeneous Packet Sizes

We still created 40 and 400 connections with uniform target connection rates 2 Gbps and 0.2 Gbps, respectively, but varied the packet sizes of connections. We used a ratio vector *pkt_size_ratio* with contents $[1, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.95]$ to set up the packet size of each connection. The packet size of the i-th connection was set to $1500 \times pkt\_size\_ratio[i \mod 10]$ bytes.

Figure 10 shows the sending rates collected by iPerf3 with heterogeneous packet sizes. The average goodputs of RPS, DBB (shared counter), and DBB (independent counter) were 18.40, 36.51, 36.46 Gbps, respectively, with 40 connections and 18.40, 36.61, 36.73 Gbps, respectively, with 400 connections. The results are also similar to those of previous experiments.



(a) (b)

Fig. 10: Goodput with heterogeneous packet sizes. (a) 40 connections (b) 400 connections

V. Conclusions

We have proposed a packet-level bandwidth-aware traffic splitting scheme which determines a set of packet dispatching rules for each switch. For each switch, the proposed scheme evenly distributes packets over all available egress links to avoid congestion on any particular link. For each stage, it also evenly distributes egress packets over all downstream switches to avoid congestion on any downstream switch. The packet dispatching rules are executed by all switches in a distributed manner. Each switch makes forwarding decisions for ingress packets without explicit probes or feedback information.

We implemented the proposed scheme on P4 switches and conducted experiments for performance measurements. The results show that the proposed scheme outperforms RPS in terms of goodput with various settings, which indicates that the proposed scheme does not suffer from packet reordering problem that arises because of imbalanced queuing build-ups in switches.

Acknowledgment

This work was supported in part by The Featured Areas Research Center Program within the Framework of the Higher Education Sprout Project by the Ministry of Education, Taiwan; in part by the Ministry of Science and Technology, Taiwan, under Grants 110-2221-E-A49-044-MY3, 110-2221-E-A49-064-MY3, 111-2218-E-011-014, and 111-2628-E-A49-019, and in part by the Ministry of Economic Affairs, Taiwan, under Grant 107-EC-17-A-02-S5-007.

References

[1] A. Singh *et al.*, "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 183–197, Oct. 2015.

[2] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1492–1525, 2018.

[3] D. Thaler and C. Hopps, "Multipath issues in unicast and multicast next-hop selection," RFC 2991, 2000.

[4] K. cheong Leung, V. O. Li, and D. Yang, "An overview of packet reordering in Transmission Control Protocol (TCP): Problems, solutions, and challenges," *IEEE Trans. Parallel Distrib. Syst.*, no. 4, pp. 522–535, Apr. 2007.

[5] P. Hurtig and A. Brunstrom, "Packet reordering in TCP," in *Proc. IEEE GLOBECOM Workshops*, Houston, TX, USA, Dec. 2011.

[6] N. M. Piratla and A. P. Jayasumana, "Reordering of packets due to multipath forwarding: An analysis," in *Proc. IEEE ICC*, Istanbul, Turkey, Jun. 2006.

[7] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 51–62, Apr. 2007.

[8] S. Prabhavat, H. Nishiyama, N. Ansari, and N. Kato, "On the performance analysis of traffic splitting on load imbalancing and packet reordering of bursty traffic," in *Proc. Int'l Conf. on Network Infrastructure and Digital Content*, Beijing, China, Nov. 2009.

[9] C. Clos, "A study of non-blocking switching networks," *Bell Labs Tech. J.*, vol. 32, no. 2, pp. 406–424, Mar. 1953.

[10] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *Proc. IEEE INFOCOM*, Turin, Italy, Jul. 2013.

[11] C. Hopps, "Analysis of an equal-cost multi-path algorithm," RFC 2992, 2000.

[12] Z. Cao, Z. Wang, and E. Zegura, "Performance of hashing-based schemes for Internet load balancing," in *Proc. IEEE INFOCOM*, Tel Aviv, Israel, Mar. 2000.

[13] Y. Kaymak and R. Rojas-Cessa, "Per-packet load balancing in data center networks," in *Proc. 36th IEEE Sarnoff Symposium*, Newark, NJ, USA, Sep. 2015.

[14] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz, "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proc. 9th ACM Conf. on Emerging Networking Experiments and Technologies*, Dec. 2013, pp. 49–60.

[15] "Intel Tofino: P4-programmable ethernet switch ASIC that delivers better performance at lower power," https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html, accessed: 2022-07-27.