Per-packet Load-balanced, Low-Latency Routing for Clos-based Data Center Networks

Jiaxin Cao[1,2*], Rui Xia[1,2*], Pengkun Yang[3*], Chuanxiong Guo[4], Guohan Lu[4], Lihua Yuan[4], Yixin Zheng[5*], Haitao Wu[1], Yongqiang Xiong[1], Dave Maltz[4]
Microsoft Research Asia[1], University of Science and Technology of China[2], University of Illinois at Urbana-Champaign[3], Microsoft[4], Tsinghua University[5],
{jiacao, v-ruxia, hwu, yqx}@microsoft.com[1],
{caojx, xiarui}@mail.ustc.edu.cn[2], pyang14@illinois.edu[3],
{chguo, gulv, lyuan, dmaltz}@microsoft.com[4], zhengyx12@mails.tsinghua.edu.cn[5]

ABSTRACT

Clos-based networks including Fat-tree and VL2 are being built in data centers, but existing per-flow based routing causes low network utilization and long latency tail. In this paper, by studying the structural properties of Fat-tree and VL2, we propose a per-packet round-robin based routing algorithm called *Digit-Reversal Bouncing (DRB)*. DRB achieves perfect packet interleaving. Our analysis and simulations show that, compared with random-based load-balancing algorithms, DRB results in smaller and bounded queues even when traffic load approaches 100%, and it uses smaller re-sequencing buffer for absorbing out-of-order packet arrivals. Our implementation demonstrates that our design can be readily implemented with commodity switches. Experiments on our testbed, a Fat-tree with 54 servers, confirm our analysis and simulations, and further show that our design handles network failures in 1-2 seconds and has the desirable graceful performance degradation property.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*

General Terms

Algorithms; Performance; Theory

Keywords

Load balance routing; low latency

*The work was performed while Jiaxin Cao, Rui Xia, Pengkun Yuan, Yixin Zheng were research interns at Microsoft Research Asia.

CoNEXT'13, December 9-12, 2013, Santa Barbara, California, USA.

http://dx.doi.org/10.1145/2535372.2535375.

1. INTRODUCTION

The IT industry is experiencing a paradigm shift by moving towards cloud computing, in which a huge amount of computing and storage resources are provided in large data centers. A large data center may contain hundreds of thousands of servers. These servers together provide various online services, e.g., Web Search, EC2 [5], Windows Azure Storage [7], and MapReduce [10]. Due to the large amount of data shuffled among the large number of servers, the network that interconnects the servers becomes a performance bottleneck. Network bandwidth is a scarce resource [10] and many latency sensitive services and applications (e.g., Web search) are experiencing long network latency tail [3].

In order to provide scalable network infrastructure, many data center network designs have been proposed [1, 12, 14, 13]. In these designs, Fat-tree [1] and VL2 [12] are built from Clos network [9], and can provide high network capacity. These Clos-based networks can be built using existing commodity Ethernet switches and routing techniques (e.g., ECMP [25] and OSPF) [20, 12]. As a result, companies are building these networks in data centers [27, 15].

Both Fat-tree and VL2 are re-arrangeable nonblocking and provide high network bandwidth. Existing routing designs [20, 12] count on ECMP for traffic load-balancing. ECMP chooses the next hop of a packet by hashing the five-tuple of the packet hence guarantees that a flow always takes the same routing path. However, due to hash collision, ECMP cannot achieve full bisectional bandwidth. As reported by [2], ECMP can only utilize 40-80% network capacity. Our measurement results further showed that even when the traffic load is moderate, network latency may have a long tail (several to tens ms, see Section 2.2). This long-tail latency causes large RTTs (e.g., more than 2ms at the 99^{th}-percentile) and flow completion time, and directly results in bad user experiences and loss of revenue [16, 18]. Flow-based improvements (e.g., Hedera [2] and MicroTE [6]) may alleviate hotspots and increase network utilization, but they cannot reduce the long latency tail, since they perform flow scheduling at seconds level.

In this paper, aiming at providing both high utilization, low latency, and short latency tail, we explore *per-packet* load-balanced routing. We design a per-packet round-robin based routing algorithm called *Digit-Reversal Bouncing (DRB)*. DRB is based on a sufficient condition which enables per-packet load-balanced routing to fully utilize network bandwidth without causing bottlenecks for both Fat-tree and VL2. In DRB, for each source-destination server pair, for each outgoing packet, the source selects one of the highest level switches as the *bouncing switch* and sends the packet to that switch. The bouncing switch then *bounces* the packet to the destination. DRB selects bouncing switches by digit-reversing their IDs, and as a result achieves perfect packet interleaving.

Compared with random-based per-packet routing, our analysis and modeling show that DRB achieves smaller and bounded queue lengths in the network. Our simulations demonstrate that DRB load-balances various traffic patterns well, results in small queue lengths and causes few out-of-order packet arrivals at the receivers. As a result, DRB achieves high bandwidth utilization (90% or more) and low network latency (795us RTT at the 99^{th}-percentile) at the same time. When integrated with TCP and ECN, DRB works for over-subscribed networks and well handles network congestions.

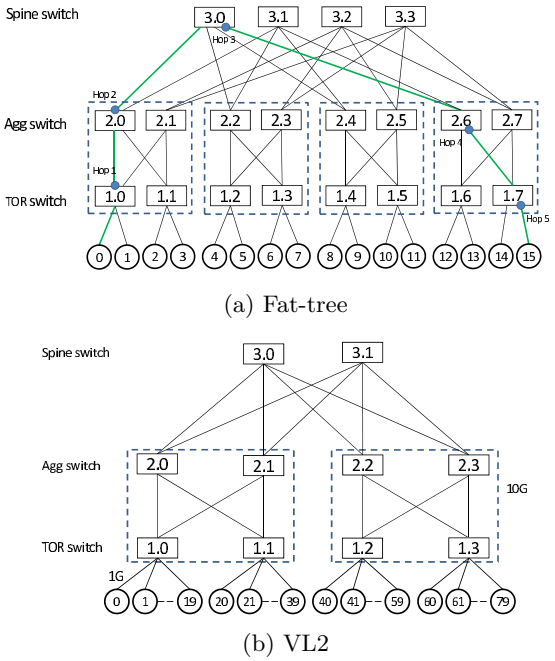
We have implemented our design with commodity switches using IP-in-IP encapsulation/decapsulation [22] for packet bouncing. We have built a Fat-tree testbed with 54 servers. Our experiments show that DRB works as designed, with average and max queue lengths of several packets and 970Mbps per-server TCP throughput, can react to network failures in 1-2 seconds, and achieves graceful performance degradation when failures happen.

The contributions of the paper are as follows. First, we design a DRB routing algorithm for load-balancing packets evenly for Clos-based networks. Second, we build an analytical model to analyze the network latency of DRB and show that DRB outperforms the other algorithms. Third, we demonstrate that DRB can be readily implemented with existing commodity switches by building a testbed.

2. BACKGROUND

2.1 Clos-based Networks

Fat-tree. Fat-tree was proposed in [1]. Fat-tree has three switch layers which are called spine, aggregation, and TOR (Top Of Rack) layers from top to bottom, respectively. All the switches have the same port number, n (an even number). Each TOR switch uses $\frac{n}{2}$ ports to connect $\frac{n}{2}$ servers, and the rest $\frac{n}{2}$ ports to connect $\frac{n}{2}$ aggregation switches. Each aggregation switch uses $\frac{n}{2}$ ports to connect $\frac{n}{2}$ TOR switches, and the rest to spine switches. In a Fat-tree, there are $\frac{n^3}{4}$ servers, $\frac{n^2}{2}$ TOR switches and aggregation switches, and $\frac{n^2}{4}$ core switches. Fig. 1(a) gives a Fat-tree network with $n = 4$.



(a) Fat-tree

(b) VL2

Figure 1: Clos-based Networks. Each dashed box contains a two-stage Clos network.

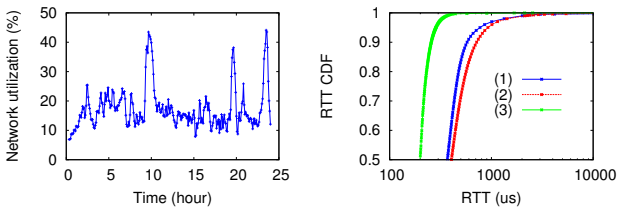
In Fat-tree, we use *bouncing switches* to do the routing. When server i sends a packet to server j, the packet first goes up to a spine switch r (up-path), and then travels down to the destination server (down-path). The spine switch is considered as a bouncing switch, since it bounces the packet from server i to server j.

Fat-tree has a nice property: *given a bouncing switch, there is one and only one path from server i to server j.* For example, in the Fat-tree shown in Fig. 1(a), given a bouncing switch 3.0, the green links show the uniquely determined path from server 0 to server 15.

VL2. VL2 [12] is also a variant of the Clos network. In the original design, VL2 uses 1GbE Ethernet for server-switch links and 10GbE Ethernet for switch-switch interconnection.

There are also three switch layers in a VL2 network. In VL2, each TOR switch has d_0 10GbE ports and $10d_0$ 1GbE ports, each aggregation switch has d_1 10GbE ports, and each spine switch has d_2 10GbE ports. Each TOR switch connects $10d_0$ servers with 1GbE ports, and d_0 aggregation switches with 10GbE ports. Each aggregation switch connects $\frac{d_1}{2}$ TOR switches and $\frac{d_1}{2}$ spine switches. In VL2, there are $\frac{d_1}{2}$ spine switches, d_2 aggregation switches, $\frac{d_1 d_2}{2d_0}$ TOR switches, and $5d_1d_2$ servers. Fig. 1(b) gives an illustration of a VL2 network with $d_0 = 2$, $d_1 = 2$, $d_2 = 4$.

In VL2, we also use bouncing switch to do the routing. However, there are d_0 paths between a server and a bouncing switch. Therefore, given a bouncing switch, the routing path is not uniquely determined. At first glance, it seems to suggest that bouncing switch based routing for Fat-tree cannot be used in VL2. In the next section, we will show that



(a) Traffic (b) RTT

Figure 2: (a) Network utilization. (b) (1) RTT CDF of light-loaded servers; (2) RTT CDF of all servers; (3) intra-rack RTT CDF.

by studying the structural property of VL2, our bouncing switch-based routing can also be used in VL2.

Fat-tree and VL2 are both Clos-based networks. They are composed of Clos networks. For example, in the Fat-tree shown in Fig. 1(a) and the VL2 shown in Fig. 1(b), each dashed box contains a two-stage Clos network. Treating each dashed box as a single virtual switch, we can see that these virtual switches are further connected by a folded Clos network.

2.2 Network Latency Measurement

Traditionally, Clos-based networks use ECMP to split traffic among multi-paths [12]. It is well-known that ECMP cannot well utilize full network bandwidth due to hash collision [2]. In what follows, we further show that ECMP may cause long network latency tail even when the network traffic is only moderate.

We have measured the RTTs and network utilization of a large data center with 94,000 servers for one day on Dec 23rd 2012. This data center supports a wide range of applications and services which include Web search and MapReduce-like computation platform. The network structure is a oversubscribed Fat-tree. The average network utilization of the third-layer switches is around 20%, with peak utilization around 45% (measured in 5min interval) as show in Fig. 2(a). As Fig. 2(a) shows, the average traffic load is only moderate.

In this experiment, we measured TCP connection setup time as RTT. All the servers participated in latency measurement and all the measurements hit the third-layer switches and have five network hops. In order to eliminate the latency introduced by busy server CPU, we filtered the servers that were busy from our result. We only counted the servers with mean CPU utilization less than 50% and the 95^{th}-percentile usage less than 75%. After server filtering, only 44,374 servers were selected. Since all the servers have 12 or more cores, our server filtering ensures that no additional latency is introduced by busy servers. After filtering, we got 4,043,399,907 RTT samples. Fig. 2(b)(1) shows the RTT CDF. Before filtering, we got 18,064,955,037 RTT samples in total. Fig. 2(b)(2) shows the total RTT CDF. We also have measured the RTT CDF of light-loaded servers in the same rack, as shown in Fig. 2(b)(3). This is to measure the latency introduced by the end-host networking stack.

Fig. 2(b)(1) shows that, though the 50^{th}-percentile (P50) and P90 RTTs are relatively small (370 us and 577 us), the P95, P99 RTTs can be as high as 772 us and 2014 us, respectively. And the curve clearly shows a long tail. Large RTTs directly result in large flow completion time and lead to loss of revenue [18].

Fig. 2(b)(2) and (3) help us confirm that the long latency tail is indeed caused by the network. Fig. 2(b)(3) shows that the intra-rack RTT is within a small range, with the P99 RTT as small as 366us. This demonstrates that the end-host stack does not introduce long latency tail. Fig. 2(b)(2) further shows that busy servers indeed increase the RTT; for example, the P90 RTT of (2) is 716us, whereas it is only 577us in (1). But busy servers do not affect the long latency tail, since the tails of (1) and (2) are almost identical.

3. DIGIT-REVERSAL BOUNCING (DRB)

3.1 DRB for Fat-tree

3.1.1 A Framework for High Network Utilization

We denote the traffic generated by the servers as an $N \times N$ matrix, where $a_{i,j}$ is the normalized traffic rate (the ratio of the traffic rate to the link capacity) from server i to j. Apparently, the normalized egress traffic rate from server i is $er_i = \sum_{j=0}^{N-1} a_{i,j}$ and the normalized ingress traffic rate to server i is $ir_i = \sum_{j=0}^{N-1} a_{j,i}$. A traffic matrix is *feasible* iff $er_i \leq 1$ and $ir_i \leq 1$.

Before proposing the framework, we first show a sufficient condition for a routing algorithm to support arbitrary feasible traffic matrix.

THEOREM 1. *In a Fat-tree network, given an arbitrary feasible traffic matrix, if a routing algorithm can evenly spread the traffic $a_{i,j}$ from server i to server j among all the possible uplinks at every layer, then all the links, including all the downlinks, are not overloaded.*

The proof is given in Appendix A. Although it looks simple, Theorem 1 gives us two important pieces of information. First, if we balance the traffic in the uplinks, the traffic in the downlinks will be balanced automatically. Second, using source-destination pair as the unit is a feasible way to achieve load-balancing.

In real world, the real-time traffic matrix is hard to obtain. Therefore, we design a traffic oblivious load-balanced routing framework for Fat-tree, which works as follows. When a source server sends a packet to a destination server, it first sends the packet to one of the bouncing switches. The packet is then *bounced* to the destination. Although this framework does not have any knowledge of the traffic matrix, it is still able to split traffic evenly in this framework. The key problem is how to select the bouncing switches properly.

3.1.2 Digit-Reversal Bouncing (DRB)

Before we present our DRB algorithm for bouncing switch selection, we introduce two algorithms, RB and RRB, and discuss their performance issues, which motivate the design of DRB.

In Random Bouncing (RB), for each packet to be sent, server i randomly chooses a bouncing switch. However, due to the randomness nature, queues can build up, which result in large network latency and large number of out-of-order packet arrivals, as we will show in Section 3.1.3.

To avoid the random burstiness caused by RB, one may consider deterministic-based Round-Robin Bouncing (RRB). In RRB, for each server pair (i, j), server i sends the first packet via a randomly selected bouncing switch $3.r$. Then RRB uses round-robin to select the next bouncing switch (i.e., the next bouncing switch is $3.\{(r+1) \mod M\}$, where M is the number of core switches).

Though RRB selects bouncing switches deterministically, it may still introduce large queues. We use an example to illustrate the problem. In Fig. 1(a), suppose server 0 is sending packets to server 15. Suppose it starts from the bouncing switch 3.0. The next bouncing switch will be 3.1. The first packet will use the up-path {0, 1.0, 2.0, 3.0}, and the second packet will use {0, 1.0, 2.0, 3.1}. Similarly, the third and fourth packets will use {0, 1.0, 2.1, 3.2} and {0, 1.0, 2.1, 3.3}, respectively. We can see that the first two successive packets both go through the link {1.0, 2.0} and the next two successive packets go through {1.0, 2.1}. In Fat-tree, the number of successive packets that go through {1.0, 2.0} is $\frac{n}{2}$. In the worst-case, when all the servers start from the same bouncing switch, the max queue length at the switches is $\frac{n}{2}(\frac{n}{2}-1)$.

To overcome the problem of RRB, we need to prevent packets of the same source-destination pair from clustering together. Using the above example, we would like to spread the packets as follows: The first packet goes through {0, 1.0, 2.0, 3.0}, and the second to fourth packets go through {0, 1.0, 2.1, 3.2}, {0, 1.0, 2.0, 3.1}, {0, 1.0, 2.1, 3.3}, respectively. Intuitively, no two successive packets should go through the same link. This is what our DRB algorithm tries to achieve.

In DRB, for each server-pair (i, j), server i sends the first packet via the switch $3.DR(r)$, where r is a randomly chosen number in $[0, M-1]$. The next packet is sent via the switch $3.DR((r+1) \mod M)$, and so on. $DR(r)$ is defined as the digit-reversal of r. Using the $\frac{n}{2}$-ary number system, r can be denoted by a two digit number, i.e., a_1a_2. Then $DR(r)$ is a_2a_1.

Using the same example in RRB, when DRB is used, the bouncing switches will be visited in the following order: 3.0, 3.2, 3.1, and 3.3. We have the following result on how DRB visits the links at different layers:

THEOREM 2. *When DRB is used for Fat-tree, the distance between any two successive visits of an i-th ($1 \le i \le 2$) layer link is $\prod_{j=1}^{i} \frac{n}{2}$.* [1]

The proof is omitted due to the space limitation. Next, we show that DRB achieves lower latencies and higher network utilization than RB and RRB.

3.1.3 Network Latency Analysis

The network latency experienced by a packet is composed of the following parts: transmission delay, propagation delay, and queueing delay. Since per-packet routing may introduce out-of-order arrivals, we need to introduce re-sequencing buffers to absorb the out-of-order arrivals for TCP at end-hosts. The re-sequencing buffers introduce additional re-sequencing delay, which is defined as the time a packet stays in the re-sequencing buffer. Since transmission and propagation delays are the same for all the algorithms, this paper focuses on queueing delay and re-sequencing delay.

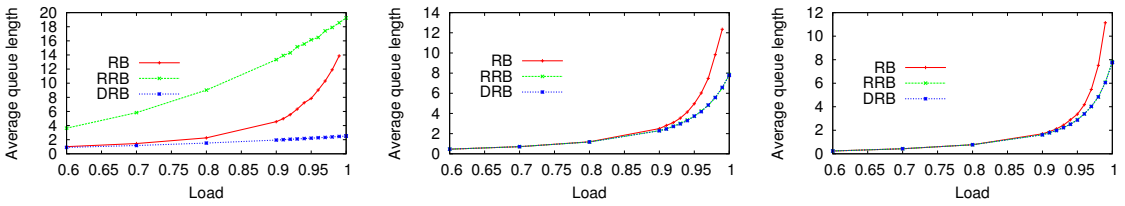
To understand the queueing delays of RB, RRB and DRB, we have built a model in Appendix B. We use permutation traffic and assume a time-slot model. In each time-slot, a server follows the Bernoulli distribution for packet generation, based on different traffic load.

Fig. 3 and Fig. 4 show the numerical results of the modeling. Fig. 3 shows the average queue length versus traffic load in a Fat-tree with all the switches having 24 ports. We have several observations: (1) the queue lengths of RB at different hops increase dramatically when the traffic load is large, while the queue lengths of both DRB and RRB are bounded even when the load is 1. (2) RRB builds up large queue length at the first hop as we have analyzed. (3) DRB performs the best. For example, its average and max queue lengths at the first hop are only 2.5 and 8, respectively.

Fig. 4 shows the queue length versus switch port number when the traffic load is 0.95. It shows that: (1) as the switch port number increases, the queue lengths at different hops increases. Except for the first hop RRB, the queue lengths converge when the port number increases. (For the first-hop queueing of RB, when n is large enough, it becomes to be the well-known $M/D/1$.) (2) DRB achieves the best performance. This is especially true for the first hop, e.g., when the port number is 24, the average 1^{st}-hop queue lengths are 16.0, 7.9, and 2.2 for RRB, RB, and DRB, respectively.

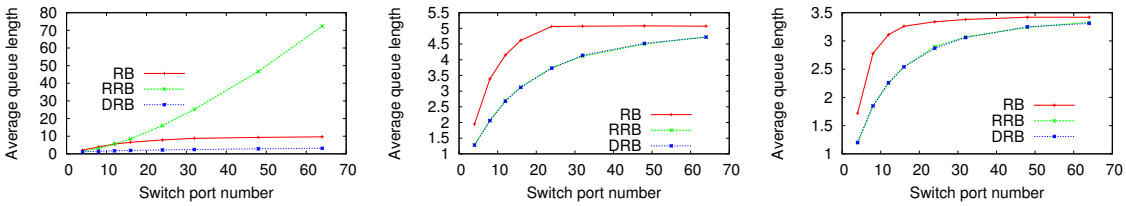
We also have calculated the queue length variances (not shown in the figures). DRB always produces the smallest queue variance along the routing paths. For example, when the traffic load is 0.95 and the port number is 24, the 1^{st}-hop variances are 151, 58, and 1.45 for RRB, RB, and DRB, respectively. Small queue variance leads to small re-sequencing delay. DRB therefore produces the smallest re-sequencing delay. Our simulation results in Section 5 further conform our analysis.

[1] If Fat-tree is extended to multiple layers, this theorem can also be easily extended.



(a) 1^{st} Hop (b) 2^{nd} Hop (c) 3^{rd} Hop

Figure 3: Average queue length vs. traffic load at hop 1, 2 and 3. The switch port number is 24.



(a) 1^{st} Hop (b) 2^{nd} Hop (c) 3^{rd} Hop

Figure 4: Average queue length vs. switch port number at hop 1, 2 and 3. The traffic load is 0.95.

Fig. 3 and Fig. 4 show the queue lengths of the up-path. Our queue modeling also explains the queueing behavior of the down-path. For Fat-tree, the 4^{th}-hop queue is similar to that of the 1^{st}-hop. The modeling results suggest that DRB achieves lower queue lengths than RB and RRB. In Sections 5 and 6, we will use simulations and experiments to demonstrate the accuracy of our modeling.

3.2 DRB for VL2

For Fat-tree, once the bouncing switch is chosen, the routing path from the source to the destination is decided without ambiguity. However, this is not the case for VL2. Using the VL2 topology in Fig. 1(b) as an example, from server 0 to switch 3.0, we have two paths. One is {0, 1.0, 2.0, 3.0} and the other is {0, 1.0, 2.1, 3.0}. Similarly, from 3.0 to server 79, we have two paths {3.0, 2.2, 1.3, 79} and {2.0, 2.3, 1.3, 79}, respectively. Therefore there are four paths from server 0 to server 79 via switch 3.0.

More generically, the number of paths between two servers via a spine switch is d_0^2. In VL2, selecting a spine switch cannot uniquely decide the routing path. Therefore, DRB cannot be directly applied to VL2. Fortunately, by studying the structure property of VL2, we can still build connections between a VL2 and a Fat-tree by introducing *virtual spine switches*.

Fig. 5 shows the extended VL2 network of the original VL2 network in Fig. 1(b). We split the physical spine switch 3.0 into two virtual spine switches 3.0 and 3.2, and the physical switch 3.1 into virtual ones 3.1 and 3.3.

For a generic VL2 network, we split each physical spine switch into d_0 virtual spine switches. For each physical spine switch i, the corresponding virtual spine switches are numbered as $\{i, i+\frac{d_1}{2}, i+\frac{2d_1}{2}, \cdots, i+\frac{(d_0-1)d_1}{2}\}$. In the original

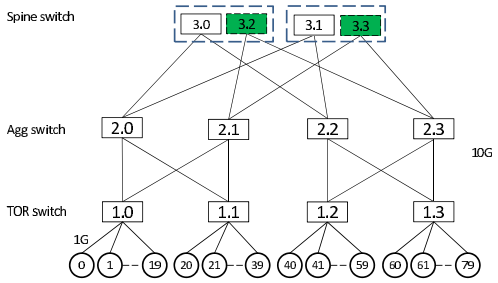


Figure 5: Virtual spine switches for VL2.

VL2 network, a spine switch has d_2 ports. In the extended network, the d_2 ports are evenly allocated to the d_0 virtual spine switches, with each virtual spine switch having $\frac{d_2}{d_0}$ ports.

The connection rule is as follows: We divide the aggregation switches into $\frac{d_2}{d_0}$ aggregation groups, each with d_0 switches. Aggregation switch i belongs to group $\lfloor \frac{i}{d_0} \rfloor$. For each group g, we number the uplink ports of this group as $p_0, p_1, \cdots, p_{\frac{d_1 d_0}{2}}$. We then connect $p_j (j \in [0, \frac{d_1 d_0}{2} - 1])$ to the g-th ($g \in [0, \frac{d_2}{d_0} - 1]$) port of the virtual switch j.

As we can see, the connection rule is the same as that of Fat-tree. It is easy to see the following theorem holds for the extended VL2 network.

THEOREM 3. *In an extended VL2 network, given a source server i and a destination server j, which are in two different aggregation groups, and a virtual spine switch r, there is one and only one shortest-path from i to j via r.*

Since Theorem 3 holds after we extend VL2 using the virtual spine switches, DRB now can be directly applied to VL2.

3.3 Oversubscribed Network and Network Congestion

Both Fat-tree and VL2 have full bisectional bandwidth and are re-arrangeable nonblocking. Theorem 1 shows that there is no congestion when the traffic matrix is feasible. In reality, congestions may occur when multiple senders send to the same receiver (i.e., many-to-one communication), or, when the network is oversubscribed (e.g., if a TOR switch in VL2 has 2 10GbE uplinks but connects 40 1GbE servers, the whole network is 2:1 oversubscribed).

DRB alone cannot handle network congestion. Since DRB already perfectly load-balances traffic in the network, the only way to handle congestion is to ask the senders to slow down. Existing TCP plus ECN can well handle this situation. When congestion is about to happen, switches in the network use ECN to mark packets, TCP as an end-host transport protocol then reacts and slows down. In Section 5, we use simulation to demonstrate that, by integrating with TCP and ECN, DRB well handles network congestion and achieves low network latency.

4. ROUTING DESIGN AND FAILURE HANDLING

We present an IP-based DRB routing design. When a source server needs to send a packet to a destination server, it first selects a bouncing switch using DRB, and then encapsulates the packet using IP-in-IP [22]. The destination address of the outer IP header is set to the address of the selected bouncing switch. When the bouncing switch receives the packet, it decapsulates the packet and sends the packet to the destination.

Our design triggers the following two design decisions: static switch routing table and server-based bouncing switch selection. We first discuss these two decisions and then discuss how we handle network failures using topology update.

4.1 Static Routing Table

When a bouncing switch is selected, there is only one path from the source to the bouncing switch and one path from the bouncing switch to the destination. Due to the uniqueness of the routing path, we therefore can configure the routing tables of the switches statically.

In our design, servers of the same TOR switch are assigned addresses in the same IP subnet and the bouncing switches are assigned /32 IP addresses. Using Fig. 1(a) as an example, switch 1.0 has routing entries on how to reach switches 3.0-3.3 (i.e., it uses uplink-port 0 to reach 3.0 and 3.1 and uplink-port 1 to reach 3.2 and 3.3). For switch 3.0, it has entries on how to reach the servers (i.e., it uses port 0 to reach servers 0-3, port 1 for servers 4-7, etc.). The number of routing entries of a switch is at most the number of server subnets plus the number of bouncing switches, which is at most thousands. Existing switches can easily support tens of thousands of routing entries, e.g, both Arista 7050 and Cisco Nexus 3000 have 16K IPv4 routes.

4.2 Server-based Bouncing Switch Selection

DRB uses server-based bouncing switch selection. DRB uses IP-in-IP packet encapsulation to select the bouncing switch. In DRB, each server needs to maintain a single integer for a machine pair's bouncing switch state. This integer is updated to the id of the next bouncing switch when a packet is sent.

This method works well when there are no switch/link failures, but failures are inevitable, and failures will affect both the up-path and down-path routings. It is possible to use the existing routing protocols to handle failures in the up-path, by leveraging ECMP and anycast techniques as shown in VL2 [12]. However, the down-path failure is much harder to handle, since there is only one down-path after the packet arrives at the bouncing switch.

We handle network failure by leveraging the rich programmability of servers. We assume that all the servers know the baseline network topology and the IP addresses of the bouncing switches. We have designed a topology update protocol to let the servers know real-time network topology, which will be discussed in the next subsection.

4.3 Topology Update

The goal of our topology update protocol is to provide servers with the real-time network topology. Topology update is essential to all routing protocols, but it is particularly challenging for us due to the following reasons: a data center may have thousands of switches, and furthermore, all the servers need to know the updated topology information in our design.

We use a switch-based proactive approach. Switches broadcast topology updates periodically or when failures are detected. We use the following rules to address the scalability issue faced by broadcast. (1) Servers do not generate or forward broadcast messages. They are passive receivers only. (2) When a TOR switch detects an uplink failure, it only delivers the message to all its servers. This reduces the number of switches that generate broadcast messages from thousands to hundreds. It works because the switches above the TORs can still detect the failures and broadcast them to the whole network. (3) When a switch receives a broadcast message from its uplink port, it forwards the message to its downlink ports. This rule further reduces the number of broadcast messages by only delivering messages along trees.

By leveraging the real-time topology information, servers evenly split traffics among all the available bouncing switches. Suppose the number of bouncing switches is n, removing one bouncing switch will only reduce $\frac{1}{n}$ network capacity. Therefore, our topology update protocol helps DRB achieve graceful performance degradation.

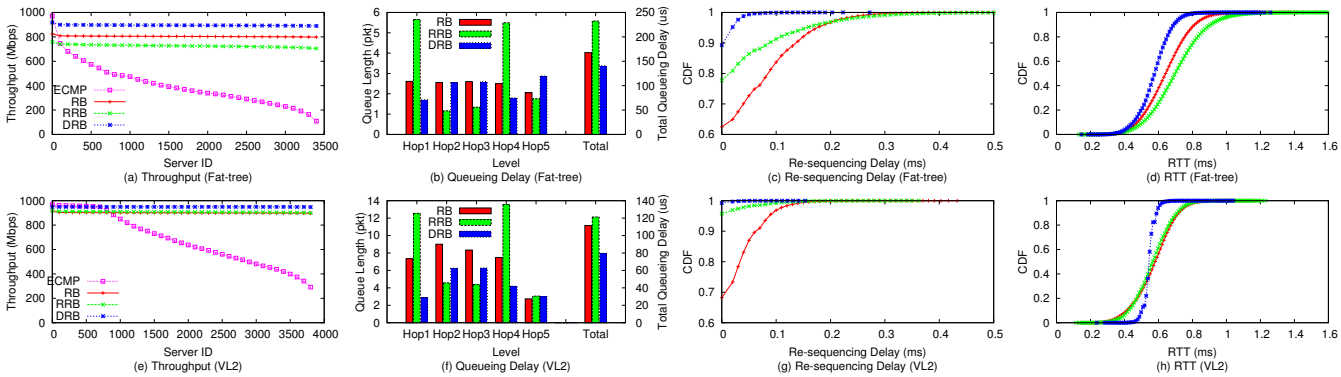


Figure 6: Performance of different routing algorithms with permutation traffic.

5. SIMULATION

In this section, we use simulations to study the performance of DRB, and compare it with RB, RRB, ECMP [12], MPTCP [23], and DeTail [28]. The network topologies we use are a Fat-tree and a VL2. For the Fat-tree, we choose $n = 24$, and the number of servers is 3456. All the links of Fat-tree are 1GbE. For the VL2, $d_0 = 4$, $d_1 = 64$, and $d_2 = 24$. VL2 uses 1GbE for server-switch connection and 5GbE for switch-switch connection, and it connects 3840 servers. The propagation delay for all the links is 5us. The buffer sizes for 1GbE and 5GbE ports are 128KB and 640KB, respectively. (We use 5GbE instead of 10GbE to let VL2 have similar server numbers as Fat-tree.)

We have implemented all these algorithms and the two topologies in NS3. We use TCP as the transport protocol. Since TCP data packets and ACKs are of different packet sizes (1500 vs 40), in our simulation and implementation, for each source-destination pair, we maintain two separate bouncing switch selectors for data packets and ACKs, respectively. We set the default TCP sending window to 256KB.

Since per-packet routing algorithms spread packets among multiple paths, they may result in out-of-order packet arrivals at the destination servers. In this paper, we use a simple re-sequencing algorithm: we buffer out-of-order packets until the sequence gaps are filled or a pre-configured time elapsed (default to 10ms).

Our simulations run at packet level, which enables us to study the fine-grained behaviors (queue length, packet out-of-order arrival, and network latency) of the algorithms. We run each simulation for one second and ignore the first 0.5 seconds for steady statistics purpose.

5.1 Permutation Traffic

In this simulation, we use a randomly generated permutation traffic pattern. In the permutation traffic, each server sends bulk data to one other server and no server receives from more than one server. Though the permutation traffic pattern is a bit artificial, it suits well for studying the throughput, network latency, and re-sequencing behaviors of different algorithms, due to its simplicity. Fig. 6 shows the result.

ECMP cannot well utilize network bandwidth. As Fig. 6(a) and 6(e) show, ECMP achieves only 396Mbps per server for Fat-tree, and 671Mbps for VL2. Our result matches that in [2]. RB, RRB, and DRB solve the flow collision problem by distributing each flow's packets into different paths. The results show that the average throughput of RB, RRB, and DRB are 803Mbps, 725Mbps, and 895Mbps in Fat-tree, and 901Mbps, 907Mbps, and 950Mbps in VL2, respectively.

DRB performs better than RB. Fig. 6(a) and 6(e) show that RB achieves lower throughput than DRB. This is because reasons as follows.

1) *RB builds up larger queues.* As we have analyzed in Section 3.1.3, RB builds up large queues when the load is high. DRB mitigates this problem by its perfect packet interleaving. As Fig. 6(b) and 6(f) show, DRB's queue buildups are lower than RB's. The 'Total' bars show the sum of queueing delays at different hops.

2) *RB introduces larger re-sequencing delays.* Since RB builds up larger queues, it causes more out-of-order packet arrivals. These out-of-order packets introduce additional re-sequencing delay. DRB causes fewer out-of-order packet arrivals, its re-sequencing delay is thus smaller. Fig. 6(c) and 6(g) shows the CDFs of re-sequencing delays. The average re-sequencing delays of RB are 38us and 17us for Fat-tree and VL2, respectively. DRB behaves much better. Its average re-sequencing delays are 2.4us and 0.2us for Fat-tree and VL2, respectively.

Since RB introduces longer queues and larger re-sequencing delays, the RTTs are apparently larger than those of DRB, as shown in Fig. 6(d) and 6(h). The 99^{th}-percentile RTTs of DRB and RB are 795us and 972us in Fat-tree, 638us and 831us in VL2, respectively. Due to its larger RTT, RB's throughput is 92Mbps lower than DRB's in Fat-tree, and 49Mbps in VL2, on average.

We also have observed that RRB achieves the lowest throughput compared with RB and DRB. The reason is that RRB builds up large queues at the 1st and 4th hops. As we show in 6(b), RRB's 1st and 4th hop queues are obviously larger

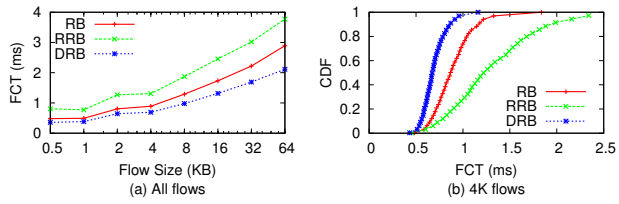


Figure 7: Flow completion time of short flows with trace-based simulation.

than those of RB and DRB, resulting in larger RTTs. Hence its throughput is lower than those of RB and DRB. However, this effect is greatly mitigated in VL2, due to the much higher uplink rate (5Gbps).

5.2 Real Trace-based Simulation

In this subsection, we use real trace to study the flow completion time (FCT) of different algorithms. The real trace was collected from a large production data center network, which contains 23k+ servers. The trace is composed of 214k+ flows. Each flow has a start time, a source, a destination, and a flow size. The trace has the following properties: 80% flows are less than 4KB and only 1% flows are larger than 50MB, and the most active servers may simultaneously communicate with 756 servers, whereas some servers simply do not talk.

We sort the servers in the trace by their sending bytes, and then map the servers into our Fat-tree. Note that by doing so, the servers in the first two racks are the busiest ones. We also reserve one server in each of the first two racks without assigning any flows. We ask the first server to send probing short flows to the second server, and measure the flow completion times. Note that in our simulation, the communications between any two racks all hit the 3rd-layer switches.

Fig. 7(a) shows the flow completion times of different per-packet load balanced routing algorithms in Fat-tree. The traffic loads of the first two racks are 92% and 90%, respectively. The results show that the FCT of DRB is the best. For example, when the flows size is 4KB, the FCT of RB is 29% higher than that of DRB (692us vs. 891us). We further plot the CDF of the FCTs when the flow size is 4KB in Fig 7(b), which illustrates that DRB's latency is consistently lower than RB's. These results also show that the FCT of RRB is the worst among the three per-packet algorithms, which agrees with the result in Section 5.1. We also have studied the FCT of ECMP. The FCT can be as large as 20.0ms for a 4KB flow, due to the network congestion caused by ECMP.

5.3 Comparison with MPTCP

We study MPTCP [23] performance using a different number of subflows. To maximize MPTCP performance, we let different subflows take different paths. Fig. 8(a,b) show the throughput and 8(c,d) show the network latency. The network latency is defined as *the time interval from the time a packet enters the first switch to the time it leaves the last switch*. The throughput result is similar to that of [23]. We have the following observations: First, the throughput of MPTCP is related to the number of subflows. The performance of MPTCP becomes better when more subflows are used. When the number of subflows is small, network utilization is low; e.g., when the number of subflows is 8, the network utilization is only 76% for Fat-tree. Second, the network latency of MPTCP is much larger than that of DRB. When the number of subflow is 32, MPTCP achieves relatively high network utilization in both Fat-tree and VL2. But the average latencies are 1154us in Fat-tree and 1090us in VL2 (or 4.4X and 6X higher than those of DRB).

DRB achieves lower latencies than MPTCP, since DRB is a proactive approach, while MPTCP is a reactive approach. DRB prevent congestion before it happens, while MPTCP handles congestion after it happens. DRB also achieves higher throughput than MPTCP, due to its lower latencies.

5.4 Comparison with DeTail

DeTail [28] reduces the flow completion time tail by using a lossless link layer priority flow control (PFC) and a per-packet adaptive load-balanced routing. When a switch forwards a packet, it randomly picks up a eligible port with the queue length smaller than the predefined threshold.

We use a Fat-tree with 8-port switches and 128 servers. We use the DeTail code provided by the authors. The DeTail code uses NSC (Network Simulator Cradle) which is time consuming. So it prevented us from using a larger network. We again use permutation traffic and TCP as the transport protocol, and study the throughput and network latency. We set up the same priority for all the flows in DeTail.

Throughput. Fig. 9(a) plots the throughput achieved by DeTail and DRB. DeTail gets 865Mbps throughput whereas DRB gets 938Mbps on average. DRB achieves 8.4% gain in throughput. Fig. 9(a) also shows that the throughput of DeTail is identical to that of RB.

Network Latency. We next compare the network latencies of DeTail and DRB. Fig. 9(b) shows the result. The average latencies of DRB and DeTail are 279us and 397us, respectively. DRB achieves not only much smaller network latency, but also shorter latency tail. The 99^{th}-percentile latency for DRB is only 440us, while it is 735us for DeTail.

The reason that DeTail achieves lower throughput and higher network latencies is that it randomly picks up a port from eligible ones. As we have analyzed in previous section, the random algorithm will cause larger queueing and re-sequencing delays. DeTail mitigates the problem by preferring the queue with the queue length smaller than the threshold. However, this reactive approach does not change the randomness nature of the algorithm. As Fig. 9 shows, DeTail has little improvement over RB.

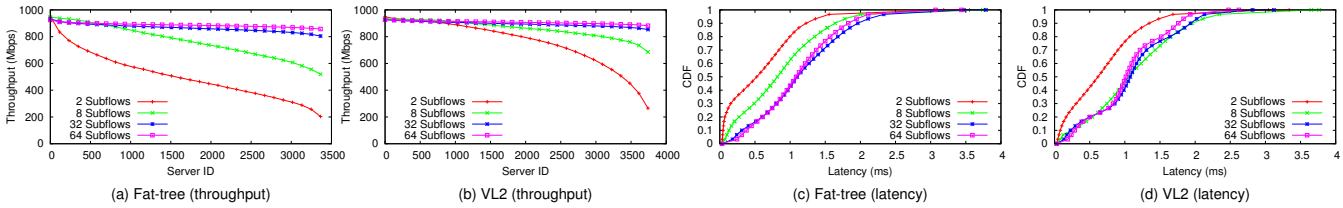


Figure 8: Performance of MPTCP.

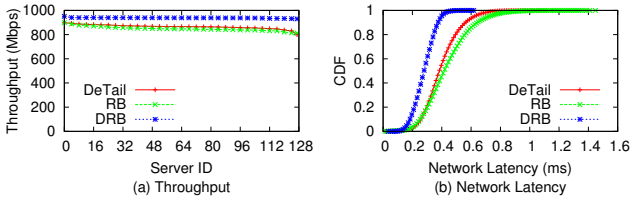


Figure 9: Comparison of DeTail and DRB.

5.5 Oversubscribed Networks

This experiment studies the performance of DRB when the networks are oversubscribed. To create 2:1 oversubscription, we double the number of servers of the first-layer switches. We use randomly generated permutation traffic, and measure both throughput and network latency. Since the network now is oversubscribed, bottlenecks will build up in the network and network latencies will increase. We then study if ECN can help to reduce latency. The ECN mark thresholds are set to 32KB and 160KB for 1GbE and 5GbE links, respectively. Similar to DCTCP, we use instant queue length.

Our findings are: (1) without ECN, all the three algorithms RB, RRB, and DRB achieve high network utilization. But the network latencies are high due to the fact that queues are built up in the network (e.g., the average network latency for DRB in Fat-tree is 839us); (2) with ECN, network latencies become much smaller (346us for DRB in Fat-tree), due to the fact that ECN can effectively control congestion based on queue length; (3) with ECN, DRB still achieves almost ideal per server throughput 456Mbps, whereas RB and RRB only achieves 418Mbps and 340Mbps in Fat-tree. The reason that RB and RRB has smaller throughput is because, as we have shown in Section 5.1, the queue lengths are larger in RB and RRB, hence ECN is triggered more frequently.

6. IMPLEMENTATION AND EXPERIMENTS

6.1 Implementation

We have implemented our load-balanced routing framework using commodity switches and servers. We first configured the static IP forwarding tables for all the switches and the IP-in-IP decapsulation at the bouncing switches. Then, we developed three software modules: (1) a kernel module at the server side that is beneath the TCP/IP stack and implements the bouncing switch selection algorithms (DRB, RRB, RB) and the IP-in-IP packet encapsulation; (2) a user space daemon at the servers that listens to topology updates from the switches; (3) a user space daemon at the switches that monitors the point-to-point links, generates link-state advertisements (LSA), and participates in LSA forwarding using the forwarding rules described in Section 4.3.

At the server side, we have implemented both the kernel and user space modules in Windows Server 2008R2. The kernel module is implemented as an intermediate NDIS (network driver interface specification) driver. The user space LSA listener collects link state broadcast messages, and then updates the network topology which is maintained in the kernel module. At the switch side, the switches run Linux, and our user space daemon runs on top of Linux.

In order to implement RRB and DRB, each server maintains the current selected spine switch for each source-destination pair with a hash table. For each out-going packet, we first find the spine switch ID in the hash table based on the source and the destination of the packet, and forward this packet to the destination with that spine switch. Then, we update the spine switch ID based on the routing algorithm, i.e., RRB or DRB. Although RRB and DRB require per-packet processing, the processing overhead is small, which is lower than 1us on average in our experiment.

We have built a Fat-tree testbed. The testbed contains 54 servers numbered from 0 to 53. The servers are 40 Dell PE R610s and 14 Dell PE2950s. The old PE2950s cannot achieve 1+1Gbps bi-directional throughput when MTU is 1.5kB. We use 9kB jumbo frame in the following experiments. Ideally, the network needs a total of 45 6-port switches with 18 at the 1st-layer, 18 at the 2nd-layer, and 9 at the 3rd-layer. In practice, we use 10 Broadcom 24-port GbE BCM956334K and 1 Quanta LB9A 48-port GbE switches. Each BCM956334K acts as 4 6-port switches and the LB9A acts as 5 6-port switches. Following the notation in Fig. 1(a), we number the 1st-layer switches as 1.0-1.17, the 2nd layer switches as 2.0-2.17, and the 3rd-layer (bouncing) switches as 3.0-3.8. The 9 bouncing switches provide 54Gbps aggregate throughput.

6.2 Experiments

Throughput. In the first experiment, we use a randomly generated permutation traffic carried by TCP. We compare the performances of DRB, RRB, and RB. For each routing algorithm, we run the experiment for 120 seconds. Overall,

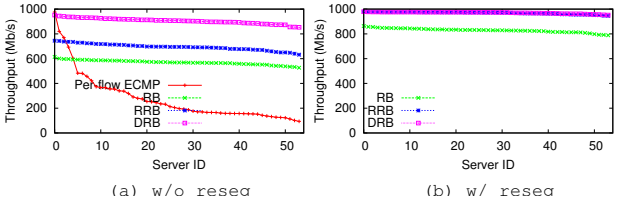


Figure 10: Throughput of the servers.

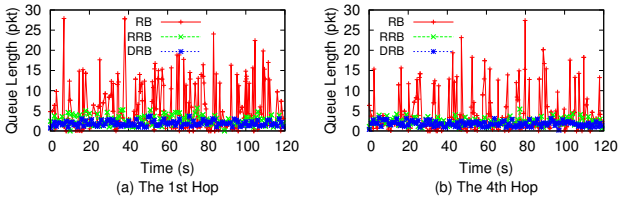


Figure 11: The queue lengths.

the experiment results echo our simulations results in Section 5 in all aspects including throughput, queue length and out-of-order arrivals.

Fig. 10(a) shows the average per-server throughput of ECMP, RB, RRB and DRB with no packet re-sequencing buffers. ECMP does not need re-sequencing buffer, but its average per-server throughput is only 279Mbps. RB, RRB, and DRB achieve 570Mbps, 694Mbps, and 905Mbps, respectively. Though DRB performs the best, the throughput is still much lower than that with re-sequence buffering (970Mbps). This experiment demonstrates the necessity of re-sequencing buffer. Fig. 10(b) also shows that RB and RRB achieves 827Mbps and 968Mbps, respectively. In this experiment, the throughput of RRB is larger than that of RB, which may seem contradict with the result in Fig. 6(a). The reason is when the switch port number is small, the performances of RRB and DRB are similar (see Fig. 4). We also have performed simulations using the same network setup and the result agrees with the experiment.

Fig. 11 plots the instantaneous queue lengths of two output ports at the 1st and 4th hops when re-sequencing is enabled at the servers. The results clearly show that the queues of RB are more burst than those of DRB; the average and max queue lengths at the 1st hop are 1.9 and 6.2 packets for DRB, 2.5 and 6.4 packets for RRB, and 3.4 and 27.9 packets for RB. Though RB achieves lower throughput, its max queue length is much larger than that of DRB. Large max queue length leads to long latency tail.

For ECMP, even though it achieves only 27.9% bandwidth utilization, it creates congested ports. In fact, we observed queue length as large as 170 packets (or 1.4MB)! Our experiment also revealed that 3%, 15% and 74% packets experience ≥ 3 out-of-order degrees in DRB, RRB and RB with re-sequencing enabled. The experiment demonstrated again that DRB achieves both high network utilization and low network latency.

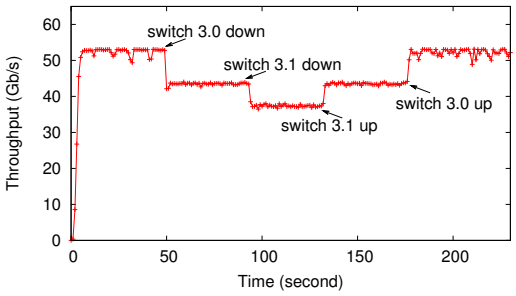


Figure 12: Failure handling and graceful performance degradation.

Failure handling. In the second experiment, we show how our routing design reacts to network failures. We use the same permutation traffic pattern and measure the aggregation throughput of all the servers. We remove the bouncing switches 3.0 at time 50s and 3.1 at 90s; then bring back 3.1 at 130s and 3.0 at 170s. We show the result in Fig. 12.

As we can see, the aggregate throughput is around 52Gbps when there is no switch failures, the throughput decreases to 44Gbps after 3.0 is down, and further decreases to 38Gbps after 3.1 is gone. We achieve graceful performance degradation in that removing one bouncing switch only reduces the capacity by 6-8Gbps. After the bouncing switches are restored, the aggregate capacity is restored accordingly. The experiment shows that the aggregate throughput becomes stable in 1-2 seconds when there are switch failures.

7. RELATED WORK

Per-packet routing has been available for Internet routers to balance traffic among multiple links for many years [8, 21]. Recently, several per-packet based designs have been proposed for data center networks. In what follows, we compare these designs with DRB.

LocalFlow [24] formulated the routing problem as multi-commodity flow, then tried to minimize the number of split flows by bin-packing multiple flows together. For flow splitting, it introduced multi-resolution splitting, which divides a flow into subflows, and a subflow has a continuous number of packets. The multi-resolution splitting is similar to RRB, hence may result in high network latency and low throughput. Furthermore, LocalFlow needs switches to perform fine-grain flow measurement and multi-resolution splitting, which cannot be done by existing commodity switches.

In [11], Random Packet Spraying (RPS) is proposed. RPS is a random based load-balancing routing algorithm, which randomly assigns packets to one of the available shortest paths to the destination. Compared with [11], we have shown that, (1) the performance of random-based routing is not as good as DRB; (2) DRB can be readily implemented using commodity switches without maintaining flow state in the switches. Furthermore, we have incorporate failure handling into our design.

In [19], simulations were used to demonstrate that random-based per-packet VLB achieves smaller latency compared with per-flow VLB, and that the performance of per-flow based, queue-length directed adaptive routing and probe-based adaptive routing can be worse than packet-level VLB. In this paper, we show, by analysis and experiments, that carefully designed deterministic DRB routing can be better than the random ones.

DeTail [28] reduces the flow completion time tail by using a lossless link layer and a per-packet adaptive load-balanced routing. We have compared DRB and DeTail in Section 5.4 and shown that DeTail may result in suboptimal throughput and network latency compared with DRB. HULL [4] also targets for high bandwidth utilization and low latency. It achieves low network latency by introducing Phantom queues, which sacrifices network bandwidth. MPTCP [23] works at transport layer. It uses the available bandwidth of a network with multiple TCP sub-flows. As we have shown in Section 5.3, though MPTCP achieves high network utilization when the number of subflows is large, it causes large queue lengths and hence high network latency. Compared with these three schemes, DRB achieves both low latency and high throughput. Furthermore, our design can be directly implemented using existing commodity switches.

Recently, D3 [26], D^{2}TCP, PDQ [17] all introduced flow-level deadline and give priority to flows that approach their deadlines. D3 and PDQ make revisions at switches and D^{2}TCP adjusts how TCP reacts to congestions. DRB works at packet-level and is complementary to these approaches.

8. CONCLUSION

Motivated by the fact that per-flow based ECMP results in both low network utilization and high network latency tail, we have presented our per-packet routing algorithm, Digit-Reversal Bouncing (DRB), for Clos-based networks. DRB interleaves the packets of a source-destination pair perfectly evenly in the network. Both our analysis and experiments show that DRB achieves higher network utilization, lower network latency, smaller latency tail, and fewer packet out-of-order arrivals than per-packet random-based routing. We have incorporated fault-tolerance in our routing design by leveraging switches for topology updates and servers for bouncing switch selection. We have built a prototype testbed and shown that DRB achieves the theoretical benefits, handles network failures in 1-2 seconds, and can be readily implemented and deployed with existing commodity network devices.

9. ACKNOWLEDGEMENT

We thank Qin Jia for her work on per-packet routing simulations and David Zats for sharing with us the DeTail code. We thank our shepherd Prof. Baochun Li and the anonymous reviewers for their valuable feedbacks on early versions of this paper.

10. REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM*, 2008.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, 2010.

[3] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *SIGCOMM*, 2010.

[4] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is More: Trading a little Bandwidth for Ultra-Low Latency in the Data Center. In *NSDI*, 2012.

[5] Amazon EC2. http://aws.amazon.com/ec2/.

[6] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *CoNEXT*, 2011.

[7] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, et al. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *SOSP*, 2011.

[8] Cisco. Per-packet load balancing. http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/pplb.html.

[9] C. Clos. A Study of Nonblocking Switching Networks. *Bell Syst. Tech. J.*, 32(2), 1953.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[11] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the Impact of Packet Spraying in Data Center Networks. In *INFOCOM*, 2013.

[12] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM*, 2009.

[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. In *SIGCOMM*, 2009.

[14] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. DCell: A Scalable and Fault Tolerant Network Structure for Data Centers. In *SIGCOMM*, 2008.

[15] J. Hamilton. 42: the answer to the ultimate question of life, the universe, and everything, Nov 2011.

[16] T. Hoff. Latency is Everywhere and it Costs You Sales - How to Crush it, July 2009. http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it.

[17] C. Hong, M. Caesar, and P. B. Godfrey. Finishing Flows Quickly with Preemptive Scheduling. In *SIGCOMM*, 2012.

[18] R. Kohavi and R. Longbotham. Online Epxeriments: Lessons Learned. *IEEE Computer*, September 2007.

[19] S. Mahapatra and X. Yuan. Load Balancing Mechanisms in Data Center Networks. In *CEWIT*, Sept 2010.

[20] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *SIGCOMM*, 2009.

[21] Juniper Networks. Overview of per-packet load balancing. http://www.juniper.net/techpubs/en_US/junos11.2/topics/concept/policy-per-packet-load-balancing-overview.html.

[22] C. Perkins. IP Encapsulation within IP, Oct 1996. RFC2003.

[23] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *SIGCOMM*, 2011.

[24] S. Sen, D. Shue, S. Ihm, and M. J. Freedman. Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing. In *CoNEXT*, 2013.

[25] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection, Nov 2000. RFC 2991.

[26] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron. Better Never than Late: Meeting Deadlines in Datacenter Networks. In *SIGCOMM*, 2011.
[27] X. Wu, D. Turner, C. Chen, D. Maltz, X. Yang, L. Yuan, and M. Zhang. NetPilot: Automating Datacenter Network Failure Mitigation. In *SIGCOMM*, 2012.
[28] D. Zats, T. Das, P. Mohan, D Borthakur, and R. Katz. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In *SIGCOMM*, 2012.

APPENDIX

A. PROOF OF THEOREM 1

PROOF. We choose one up-path and one down-path to show that they are not overloaded. The up-path we choose connects server 0 to a core switch. We number the contained links as l_0, l_1 and l_2. Similarly, the down-path connects a core switch to server $N-1$, and we number the links as l_3, l_4 and l_5.

It is easy to see that l_0 is not overloaded, since the traffic over l_0 is $\sum_{i=0}^{N} a_{0,i} \leq 1$. Similarly, link l_5 is not overloaded. For uplink $l_m (1 \leq m \leq 2)$, it carries traffic for server pairs (i,j), $i \in [0, M-1]$ and $j \in [0, N-1]$, where $M = \prod_{i=1}^{m} \frac{n}{2}$ and N is the total number of servers. We know that the traffic from server i to j is $a_{i,j}$, and the portion sent to l_m is $\frac{1}{M} a_{i,j}$. The traffic carried by l_m therefore is

$$\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (\frac{1}{M} a_{i,j}) = \sum_{i=0}^{M-1} \frac{1}{M} \sum_{j=0}^{N-1} a_{i,j} \leq \sum_{i=0}^{M-1} \frac{1}{M} = 1.$$

Similarly, for downlink $l_{m'} (3 \leq m' \leq 4)$, it carries traffic for server pairs (i,j), $i \in [0, N-1]$ and $j \in [N-M', N-1]$, where $M' = \prod_{i=1}^{5-m'} \frac{n}{2}$. When a packet hits a switch at the highest level, the path from that switch to the destination server is uniquely determined. Hence the portion of traffic carried by $l_{m'}$ for (i,j) is $\frac{1}{M'} a_{i,j}$. The total traffic carried by $l_{m'}$ therefore is

$$\sum_{i=0}^{N-1} \sum_{j=0}^{M'-1} (\frac{1}{M'} a_{i,j}) = \sum_{j=0}^{M'-1} \frac{1}{M'} \sum_{i=0}^{N-1} a_{i,j} \leq \sum_{j=0}^{M'-1} \frac{1}{M'} = 1.$$

We therefore prove that all these six links are not overloaded. Similarly, we can prove that all the rest links are not overloaded. □

B. QUEUE MODELING

We use modeling to understand the queueing behaviors of DRB, RB, and RRB at different hops in a Fat-tree. We assume all the switches are output queued, the capacities of all the ports are 1, the packets are of the same length and one output switch port transmits one packet in one time-slot if its queue is not empty.

The queue at an output port can be described by

$$q(t+1) = (q(t) - 1)^+ + \sum_i f_i(t+1)$$

where $q(t)$ is the queue length at time-slot t and $(q(t)-1)^+ = 0$ when $q(t) = 0$. Once the set of input flows f_i is decided, the queue and the output can all be calculated numerically.

To make the modeling tractable, we assume permutation traffic pattern. In each time-slot, each input generates packets using the Bernoulli distribution. Though the real-world traffic pattern is more complicated (and much harder for analysis), the analysis of permutation traffic pattern can shed light in understanding the performances of different algorithms.

Our modeling focuses on Fat-tree. Modeling for VL2 can be performed similarly. As we have shown in Fig. 1(a), there are 5 hops in a Fat-tree. We show how to model queues at these five hops one-by-one.

Queue modeling of the up-path hops. A first-hop queue is modeled as one server with one output flow and $\frac{n}{2}$ input flows, and the average throughput of every input flow is $\frac{2\rho}{n}$, where $\rho \leq 1$ is the server input load.

A second-hop queue can be modeled as one server with one output flow and $\frac{n}{2}$ input flows, and each input flow in turn is the output flow of a first-hop queue with $\frac{n}{2}$ input flows. The average throughput of a first-hop input flow is $\frac{4\rho}{n^2}$.

Queue modeling at the bouncing switches. To model a third-hop queue, we observe that the number of destination flows of the output of a third-hop queue is $\frac{n^2}{4}$, and these destination flows come from n input ports of a layer-3 switch. Since a permutation is generated randomly, we therefore can randomly assign these $\frac{n^2}{4}$ flows to the n input ports. The input of a layer-3 port in turn is the output of a second-hop queue.

The modeling differences of RB, RRB and DRB are that they produce different input flows. For the first-hop queue modeling, the traffic for an input flow is generated as follows. For RB, in each time-slot, a packet is generated with probability $\frac{2\rho}{n}$. For RRB, in each time-slot, a packet is generated with probability ρ, and then $\frac{n}{2}$ successive packets are sent to an input flow for every $\frac{n^2}{4}$ generated packets. For DRB, in each time-slot, a packet is generated with probability ρ and one packet is sent to an input flow for every $\frac{n}{2}$ generated packets.

For the second and third-hop queue modeling, packets of all the input flows in RB are still randomly generated, with probability $\frac{4\rho}{n^2}$, whereas in RRB and DRB, one packet is sent to an input flow for every $\frac{n^2}{4}$ generated packets.

Queue modeling of the down-path hops. The queues of the down-path hops are more difficult to model due to the 'network' effect, where other 'interference' flows may change the packet arrival patterns of the input flows of a queue. Nonetheless, we can still approximate the queueing behavior by omitting the interferences of the previous hops. By ignoring the interferences introduced by the network, it is easy to see that hop 4 is symmetric to hop 1. Therefore they have similar queue lengths.