# Hashing Design in Modern Networks: Challenges and Mitigation Techniques

*Yunhong Xu*[†], *Keqiang He*[‡], *Rui Wang*[‡], *Minlan Yu*[§‡], *Nick Duffield*[†],
*Hassan Wassel*[‡], *Shidong Zhang*[‡], *Leon Poutievski*[‡], *Junlan Zhou*[‡], *Amin Vahdat*[‡]
[†]*Texas A&M University* [§]*Harvard University* [‡]*Google, Inc.*

## Abstract

Traffic load balancing across multiple paths is a critical task for modern networks to reduce network congestion and improve network efficiency. Hashing which is the foundation of traffic load balancing still faces practical challenges. The key problem is there is a growing need for more hash functions because networks are getting larger with more switches, more stages, and increased path diversity. Meanwhile, topology and routing become more agile in order to efficiently serve traffic demands with stricter throughput and latency SLAs. On the other hand, current generation switch chips only provide a limited number of uncorrelated hash functions. We first demonstrate why the limited number of hashing functions is a practical challenge in today's datacenter network (DCN) and wide-area network (WAN) designs. Then, to mitigate the problem, we propose a novel approach named *color recombining* which enables hash functions to reuse via leveraging topology traits of multi-stage DCN networks. We also describe a novel framework based on *coprime* theory to mitigate hash correlation in generic mesh topologies (i.e., spineless DCN and WAN). Our evaluation using real network trace data and topologies demonstrate that we can reduce the extent of load imbalance (measured by the coefficient of variation) by an order of magnitude.

## 1 Introduction

Traffic load balancing is critical to the reliability and efficiency of modern datacenter and wide-area networks [16, 34, 37]. One widely deployed technique for traffic load balancing is Equal-Cost Multi-Path (ECMP) routing [16], where packets forwarding to a destination are load-balanced over multiple paths based on hashing of the packet header that takes place in switch hardware. ECMP and its variant Weighted-Cost Multi-Path (WCMP) [37] allow proper utilization across abundant paths available in modern networks. Hashing on header fields allows packets of the same flow to follow the same path without incurring packet reordering. As ECMP/WCMP offers a number of nice properties, including stateless operation and no reordering, it is the de facto standard for traffic load balancing in large IP networks [1, 28, 37].

Switch hashing is the foundation of traffic load balancing in modern multi-path networks. To optimally load balance traffic and utilize full available bandwidth in multi-path networks using ECMP/WCMP, hash function allocation across a network should adhere to the following rule: *no correlated hash functions should appear on the same forwarding path in the absence of color recombining* (§3). Unfortunately, commodity switch chips only support a limited number of hash functions. For example, the software development kit for Broadcom switches supports *RTAG*7 [9], a hashing scheme utilizing seven hash functions; the Cisco Nexus 5500 Series offers eight versions of *CRC*8 [8]; the switches deployed in our datacenters have six independent hash functions. It is difficult to implement a large set of complex hash functions because hash computation becomes a bottleneck at high line rates [17, 19] and switch chip architects often need to trade-off between high line rate and hash function complexity.

Because of the limited number of hash functions provided in switch chips, a challenge in network design is that the reuse of correlated or even identical hash functions in different different switches along the same end-to-end path for a flow causes *traffic polarization and inherent load imbalance*. Many network operators have observed this problem before [7, 11, 14, 18, 25]. One example is the Cisco Express Forwarding (CEF) Polarization phenomenon [7], where different switches repeatedly use the same hashing algorithm, resulting in a switch selecting a small portion of links for all traffic destined for one prefix, while other links were underutilized. Another example is the hashing imperfection problem observed by a large cloud provider [18, 25], where correlated hash functions lead to network congestion and even high priority traffic losses which directly impact application performance. In this paper, we use the term *hash correlation* to describe the association between hash functions in different switches that leads to traffic polarization.

Researchers and network operators have proposed a few

approaches to mitigate hash correlations. One class of methods uses variations of available hash functions. The most common approach uses hash functions with different seeds to avoid hash correlation. However, contrary to conventional wisdom, seeds are not as effective as expected as we will show theoretically and experimentally in this paper. Another approach uses Time-To-Live (TTL) in the packet header for hashing. There are two ways of using TTL: a) using TTL as part of hashing input, which has the following limitations in production: 1) it breaks traceroute because probe packets are hashed/pathed differently at the same hop; 2) IP in IP tunneling [12] (a common technique for network virtualization in the cloud) commonly uses the inner IP headers because they have more entropy than the outer IP headers, but the inner headers' TTL does not change. It's also hard to select outer header TTL and inner headers because switches do not have enough bit vectors. b) choosing a different hash function based on TTL [14], in addition to 1) and 2) mentioned in a), the challenges are: 3) it also requires modifying switch hardware, which presents administrative and commercial barriers to implementation since switch chips deployed in most datacenters do not support this operation currently; and 4) it is still constrained by the limited number of hash functions implemented in ASIC.

This paper focuses on fixed-function switches which are still the majority of devices in datacenters in the industry today. Hyperscalers cannot replace all switches with programmable ones in the near future. In this paper, we first demonstrate that the limited number of hash functions in commodity switch chips poses a practical challenge in modern DCNs and WANs and hinders the development of more advanced network topology and routing designs. To overcome this challenge, we propose two novel techniques that improve hashing, the cornerstone of traffic load balancing, substantially while respecting the limited number of independent hash functions available in commodity switches– 1) for widely adopted, multi-stage Clos DCNs, we propose a *color recombining* approach by comparing the effect of hash functions on traffic to light passing through multiple triangular prisms: the color recombining technique leverages the trait of multi-stage Clos topology where polarized traffic gets recombined to *color white* after passing through a multi-stage Clos. Color recombining enables us to reuse certain hash functions along the forwarding path without causing hash correlations. We discuss hashing design for one of the multi-stage Clos DCNs, Jupiter [26], and how the color recombining technique helps reduce the required number of independent hash functions; 2) for non-hierarchical mesh networks such as spineless DCN [13,32] and WAN, we propose a novel framework residing in the Software Defined Networking (SDN) [20] controller. The framework mitigates the effect of hash correlations by the selection of the divisors $n$ ($n$ is also known as group size in ECMP/WCMP routing) used to map a flow's hash value $h$ to the output port over which a packet is forwarded, i.e., $h\%n$. Specifically, we establish that *when ECMP/WCMP group sizes at different switches are coprime, then the effects of underlying correlations between hash functions are reduced significantly*. We add a small amount of logic to the SDN controller to ensure the group sizes of switches with correlated hash functions are coprime. As a software-only approach, this coprime-based approach is compatible with commodity switch chips.

To summarize, the contributions of our work are as follows:

1. Details the hashing design in *multi-stage Clos DCN* and proposes a *color recombining* method to allow hash function reuse and to reduce the number of hash functions needed in multi-stage Clos DCNs;

2. Identifies an approach based on the *coprime* theory to mitigate hash correlations for generic *mesh networks such as spineless DCN and WAN* that require a large number of independent hash functions, and proposes algorithms to coprime group sizes for both ECMP and WCMP routing;

3. Evaluation results based on real network trace data and topologies demonstrate color recombining and coprime techniques' effectiveness in mitigating hash correlation – they can reduce the extent of load imbalance (measured by the coefficient of variation, CV) by approximately an order of magnitude.

The color-recombining approach uses topology and forwarding structures that are common in Clos networks and requires no hardware or software changes except hash function reconfiguration. The coprime-based method works with any topologies (e.g., mesh networks) but it requires controller software changes and only resolves polarization when the conditions described in §4 are met.

In addition to the technical contributions, we also would like to call for switch vendors' attention to offer better hashing support in future generations of chips to facilitate flexible network designs.

## 2 Background and Motivation

In this section, we first motivate why hashing is an important problem in modern DCNs and WANs. Then, we provide the background of ECMP and WCMP. Finally, we introduce the traffic polarization issue caused by hash correlation and reveal the fact that the current generation of switch chips only provides a limited number of independent hash functions which poses a practical challenge to traffic load balancing in modern networks.

The implications of bad hashing are twofold. First, bad hashing leads to *traffic polarization* that endangers reliability (due to reduced path diversity), wastes network bandwidth, cancels efficiency gains of traffic engineering, and inevitably

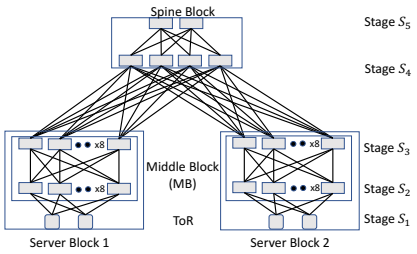



Figure 1: Illustration of Jupiter DCN (5-stage Clos). For simplicity, only one *Middle Block* of each *Server Block* and one *Spine Block* of the spine layer is shown.

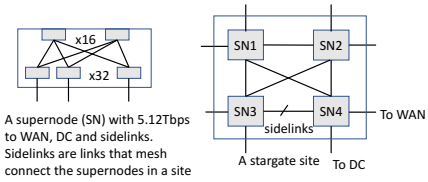



Figure 2: Illustration of a *Stargate* site in the mesh-connected B4 WAN. Figure is adapted from [15].

increases network cost. Second, bad hashing causes *inherent traffic load imbalance* and leads to network congestion that affects application performance.

## 2.1 Hashing is a Practical Challenge in Network Designs

There are several trends that make hashing an increasingly important problem in modern networks: 1) both datacenter and wide-area networks are getting bigger with more switches and stages; 2) modern networks are very dense and have a large number of paths between any two nodes; 3) topology and routing become more agile and flexible in order to improve network efficiency and availability, e.g., the move from spinefull DCN to reconfigurable spineless [13, 32] and the use of non-shortest path routing to improve availability and performance [15]; and 4) emerging applications (such as distributed machine learning) are becoming more throughput hungry while demanding stricter network SLA guarantees. These new trends pose challenges to commodity switches' hashing capability, which is the cornerstone of traffic load balancing.

### 2.1.1 Multi-stage Clos DCN

Modern DCN connects a massive amount of compute/storage nodes, runs critical services, such as Search Serving, Video Serving, Geo & Map, Cloud, and Gaming, etc, and has very large path diversity; therefore, hashing and traffic load balancing are crucial. We use the Jupiter topology [26, 35] (as illustrated in Figure 1) as the case study of multi-stage Clos DCNs. We denote a *Server Block* (aka Pod) as *SB* and the

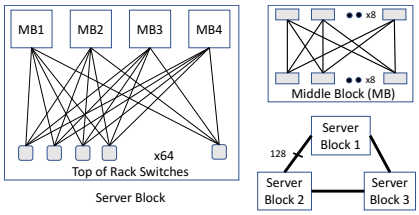



Figure 3: Spineless DCN [32] to simulate hash correlation's impact on traffic load balancing.

5 switch stages as $S_1$ to $S_5$. So the longest forwarding path is when a packet is routed from a ToR in $SB_i$ to another ToR in $SB_j$ and the hop sequence is $S_1(SB_i) \rightarrow S_2(SB_i) \rightarrow S_3(SB_i) \rightarrow S_4 \rightarrow S_5 \rightarrow S_4 \rightarrow S_3(SB_j) \rightarrow S_2(SB_j) \rightarrow S_1(SB_j)$ where $i, j$ are server block indices. To achieve optimal load balancing performance, hashing needs to follow the following property: no correlated hash functions should appear on the same forwarding path. Naively we need $O(2L)$ (more precisely, $max\_number\_of\_hops - 1$) independent hash functions, where $L$ is the number of layers (or stages) of the fabric. In the case of Jupiter, 8 hash functions are needed. Unfortunately, there are only 6 uncorrelated hash functions provided by the switch chips deployed in our datacenters and the requirement of 8 independent hash functions already makes the commodity switch's hashing capacity stretched. We will discuss how we reduce the number of hash functions needed for multi-stage Clos DCN in Section 3.

### 2.1.2 Spineless DCN

Recently, there are new *spineless* DCN topologies [13, 32] which reduce cost and enable faster tech refresh, but require more hash functions. Figure 3 is an example of spineless DCN topology. In spineless DCN, server blocks are directly connected via a mesh and the spine blocks are completely removed to reduce network cost (including the cost of both spine switches and the associated optics) significantly and to enable faster switch generation evolvement. Also, non-shortest path routing is used to improve routing path diversity for high availability and enable traffic engineering to optimize network link utilization.

While spineless DCNs are cost-effective and efficient, hashing design becomes more challenging because the number of hash functions required depends on the number of server blocks instead of the number of layers as in multi-stage DCNs. Taking the Gemini [32] spineless DCN as an example, assuming that we only allow at most one transit server block in routing, then the longest forwarding path of a packet is: $S_1(SB_i) \rightarrow S_2(SB_i) \rightarrow S_3(SB_i) \rightarrow S_3(SB_j) \rightarrow S_2(SB_j) \rightarrow S_3(SB_j) \rightarrow S_3(SB_k) \rightarrow S_2(SB_k) \rightarrow S_1(SB_k)$ where $i, j, k$ are the indices of three randomly chosen server blocks. The key challenge is that we need to make sure there are no correlated hash functions for any $i, j, k$ combinations. So spineless DCN requires $O(N)$ hash functions where $N$ is the number of server

blocks. $N$ ranges from 10s to 100s in a typical fabric, so it is very challenging to design hashing with the limited hash functions provided by current-generation switch chips.

#### 2.1.3 WAN

Traffic engineering and load balancing are critical to improving WAN's performance and reducing operational costs. Mesh-connected WANs have the same hashing design challenge where the number of uncorrelated hash functions required is subject to the scale of the network. For example, Figure 2 shows the topology of a Stargate site of the B4 WAN [15]. Each site is composed of up to 4 supernodes where a supernode is a 2-stage Clos with links to WAN, Data Center, and other supernodes in the same site. B4 site-level topology is a partially connected mesh and non-shortest path routing is employed for both availability and efficiency (i.e., traffic engineering) purposes. Similar to spineless DCNs, to ensure optimal load balancing performance, we need $O(N)$ hash functions, where $N$ is the number of sites in the WAN. Note that B4 grew 7× larger from 2012 to 2017 [15]. We will discuss the hash correlation mitigation technique for mesh networks such as spineless DCNs and WANs in Section 4.

### 2.2 ECMP/WCMP Traffic Load Balancing

Equal-Cost Multiple-Path (ECMP) [16] – a routing and traffic load balancing strategy that allows traffic between a source and destination node to be transmitted across multiple paths – identifies a set of routes, each of which is equal-cost towards the destination. The routes identified are referred to as an *ECMP group*. An ECMP group is defined at flow-level. When forwarding a packet, the routing strategy decides which next-hop path to use based on a hashing algorithm. That is, the route of a packet is determined by the mapping from the hash value to an egress port, i.e., $h \% n$, where $h$ is the hash value, and $n$ is the number of output ports in the ECMP group. The typical IP packet header fields used for hashing input are: source IP, destination IP, transport protocol, TCP/UDP ports, and IPv6 flow label.

Each route in an ECMP group has an equal chance for traffic forwarding. For example, Figure 4a shows the traffic from source IP prefix to destination IP prefix uses four equal-cost paths, which are labeled in four colors, where $H_1$ and $H_2$ are two independent hash functions.

When switch hashes packets across multiple paths according to customized weights instead of uniform ones, this variant of ECMP is called Weighted-Cost Multi-Pathing (WCMP) [37]. And the set of routes identified for a flow is referred to as a WCMP group. WCMP can be implemented via replicating ECMP table entries in the switch to approximate the intended WCMP weights. For example, if there are 2 output ports of a flow $f$, denoted by $p1$, $p2$, and the intended weights are 2:1, then the WCMP group can be implemented as $p1$, $p1$, $p2$ ($p1$ is duplicated as 2 entries to achieve the 2:1 traffic split).

### 2.3 Hash Correlation Causes Traffic Polarization and Load Imbalance

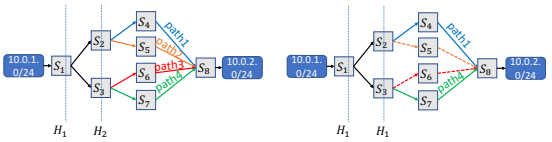
#### 2.3.1 Limited Number of Hash Functions Leads to Hash Correlation

ECMP/WCMP is widely deployed in modern DCNs and WANs which have a large path diversity to improve traffic load balancing performance and reduce network congestion. Switches are configured with hash functions that compute hash values based on packet headers and forward packets via selecting one out of multiple next-hops based on the hash value and the ECMP/WCMP group size.

However, current-generation switch chips were designed for small-scale DCNs or large but sparse ISP networks and only provide a handful of independent hash functions. For example, the switch chips used in our datacenters provide six independent hash functions. The software development kit for Broadcom switches supports RTAG7 [9], a hashing scheme utilizing seven hash functions. The Cisco Nexus 5500 Series offers eight versions of *CRC*8 [8]. CRC and XOR are two popular hashing algorithms because these two algorithms have been used in communication systems with mature and efficient ASIC implementation, which includes plenty of circuit optimization [33]. As discussed in Section 2.1, the number of independent hash functions needed is far beyond (orders of magnitude difference) what is provided today, especially for spineless DCN and WAN.

In fact, it is difficult to implement a large set of uncorrelated hash functions because complex hash functions become a bottleneck at high line rates [17, 19]. For example, the authors in [19] discovered that the generation of Cyclic Redundancy Codes (CRCs) represents the main bottlenecks in iSCSI protocol processing. Switch chip architects often need to trade-off between high line rate and hash function complexity and we are not aware of any switch chips with cryptographic hash functions today due to computation complexity concerns.

In an ideal world where all the switches on the forwarding path are configured with completely independent hash functions, there would be no hash correlation. Due to the limited hash functions, we have to reuse them, which leads to hash correlation and traffic polarization. *Polarization* is a term used to describe what happens to light as it travels through a filter. Only light rays that have a certain characteristic get through the filter. We can take the same term and apply it to network traffic. Traffic polarization is the effect when a set of packets choose a particular path and the redundant paths remain completely unused [7]. Traffic polarization reduces path diversity and causes sub-optimal use of redundant paths and results in traffic load imbalance and network congestion. For example, Figure 4b shows that switches $s_1$, $s_2$ and $s_3$ employ the same




(a) Illustration example of ECMP.

(b) ECMP with hash correlation.

Figure 4: ECMP with and without hash correlation. In (b), two out of four paths are unused due to hash correlation.

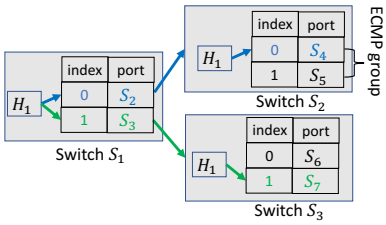



Figure 5: Hash correlation leads to correlated load balancing decisions.

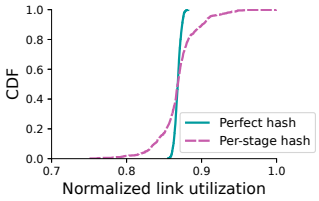



Figure 6: Normalized link utilization for perfect hashing and per-stage hashing

hash function ($H_1$). The hash correlation between switches $s_2$ ($s_3$) and $s_1$ causes traffic polarization. Figure 5 explains the problem: hash value on $s_2$ ($s_3$) is the same as $s_1$, and thus flows are only forwarded to $s_4$ from $s_2$ and $s_7$ from $s_3$. As a result, traffic from source to destination only uses two routing paths instead of four as in Figure 4a. Traffic polarization endangers reliability, wastes network capacity, and leads to hot links and network congestion under high traffic loads.

### 2.3.2 Random Seeds Are Not Effective

Since the shortage of independent hash functions available in switches prevents simply assigning independent hash functions to different switches, switch vendors suggest deriving multiple hash functions from one via using a switch-specific seed [7]: a seed is an initial value to start the CRC computation via XORing the input data. Because switch chip's port count is powers of 2 and ports are typically divided into two equal-size directions (e.g., up-facing and down-facing) in modern networks, ECMP groups with an even number of ports are prevailing. However, we found that seeds do not work for an ECMP group of an even number of ports. For example, based on Theorem 1, all packets on switch $s_2$ (Figure 4b) from switch $s_1$ have the same routing choice even if they use different seeds for the same CRC; the proof of Theorem 1 can be found in the appendix.

**Theorem 1** *If* $crc_b(x \oplus z_1)\%2 = crc_b(y \oplus z_1)\%2$, $crc_b(x \oplus z_2)\%2 = crc_b(y \oplus z_2)\%2$, *where* $x$ *and* $y$ *denote the data with the same size in bytes,* $z_1$ *and* $z_2$ *are two seeds of* $b$ *bits, and* $b$ *is the integer to denote the number of bits of the CRC, and* $\oplus$ *denotes eXclusive OR (XOR).*

The above theorem indicates that choosing two random seeds does not create two independent hash functions from one *CRC* polynomial. Please note CRC polynomials are the most widely implemented hash functions in switch hardware, e.g., Broadcom's RTAG7 is a hash family with 6 *CRC*16s and 1 *CRC*32. Applying a random seed is a linear operation and it can not decorrelate a hash function's output effectively.

To confirm our theoretical analysis, we also build a spineless DCN topology (shown in Figure 3) to simulate traffic load balancing performance with hash functions provided by a switch vendor. The topology is composed of three *Server Blocks* and each server block contains 64 ToRs (each ToR is assigned a /24 IP prefix). All these three server blocks have the same radix of 256. The three server blocks are connected in a full mesh (i.e., this is a spineless DCN [32]), so there are exactly 128 links between each server block pair. To simplify the analysis, we generate flows (the source IP and destination IP of a flow is randomly chosen from the source ToR and destination ToR's IP prefix range respectively) following a uniform traffic pattern, i.e., all the server block pairs have the same amount of flows and the flow size distribution follows empirical datacenter flow size measurement in [4].

We compare two hashing schemes: one is *perfect hashing*[1] on each switch; and the other is *per-stage hashing with random seed* where the switches in different stages are configured with different functions provided by the switch vendor and each switch in the network is provided a completely independent and random seed. We measure the amount of traffic landed on the links between server blocks and show the normalized link utilization distribution in Figure 6. We can see from Figure 6 that the link utilization is close to uniform when using perfect hashing (which is expected) while per-stage hashing with random seeds leads to considerable traffic imbalance (max/min link utilization is 1.33). This experiment confirms that random seed does not solve the hash correlation problem.

## 3 Hashing Design in Multi-stage Networks

In this section, we describe the hashing design for multi-stage Clos DCNs. We first start with a strawman solution, *per-stage hashing with random seed*, then introduce *per-port hashing* scheme which requires more hashing functions than what is provided by commodity switches. In order to reduce the number of independent hash functions needed, we propose a novel approach named *color recombining* by leveraging topology traits of multi-stage Clos networks where polarized traffic is recombined into non polarized traffic and hash function reuse is allowed. We use Jupiter topology [26] as the case study here but the techniques proposed in this section can

[1] Using the Mersenne Twister pseudo-random number generator in C++.

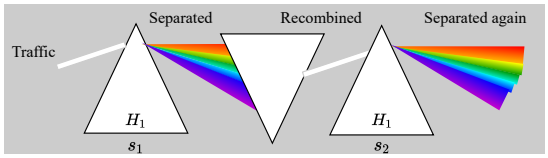



Figure 7: Illustration of color recombining concept.

be generalized to other multi-stage Clos DCNs. At the end of this section, we will provide details on why the hashing design for multi-stage Clos does not work in spineless DCNs and WANs.

One naive hashing design for Jupiter is *per-stage hashing with random seed* which means we assign a different hash function to each stage of switches and each switch is fed with a random seed. However, as we revealed in §2, random seed does not solve the hash correlation problem. Instead, we propose *per-port hashing* which means we apply hash function based on the input port of the switch. In total, there are 5 stages of switches in Jupiter as shown in Figure 1 and we apply two hash functions (one for upward traffic and the other for downward traffic) per switch except $S_1$ and $S_5$. $S_1$ is ToR so only upward traffic requires a hash function; there is no need to have two hash functions in $S_5$ because we do not need to distinguish upward and downward traffic. Therefore, we need 8 independent hash functions to implement per-port hashing in Jupiter. And we can prove that for any routing path, there is no hash correlation. Taking the longest forwarding path as an example: a flow is routed from a ToR in the source server block to another ToR in the destination server block. So the forwarding path is $S_1(source\ ToR) \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_4 \rightarrow S_3 \rightarrow S_2 \rightarrow S_1(destination\ ToR)$. At each hop, a unique hash function is used so there is no hash correlation.

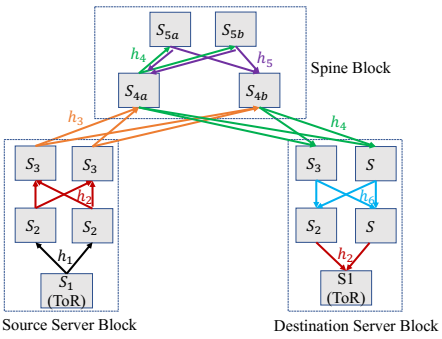



Figure 8: Per-port hashing with color recombining in Jupiter.

Per-port hashing design is clean and elegant (i.e., hash function allocation is static and easy to configure) for multi-stage Clos networks such as Jupiter, but the number of hash functions required is larger than what is provided by commodity switch chips. To reduce the number of hash function needed by per-port hashing, we identify and propose the *color recombining* technique via exploiting topology properties of

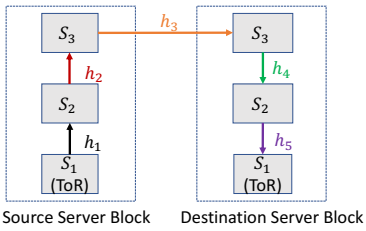



Figure 9: Per-port hashing works if only direct path (i.e., shortest path) routing is allowed in spineless DCN.

multi-stage Clos to enable hash function reuse. We compare traffic going through a hash function to light passing through a triangular prism: light passes through prism and gets separated to its component colors; analogously, network traffic goes through the hash function and gets forwarded to multiple paths. For example, Figure 7 shows that $H_1$ can be reused in switch $s_2$ because traffic becomes *color white* after passing through the middle triangular prism, where color white denotes the combined traffic before splitting or after recombining. We use the term *color recombining* to denote the recombining of polarized/dispersed traffic. Our key insight is that *in multi-stage Clos networks, certain stages of switches serve as the middle triangular prism in Figure 7 and combine polarized traffic to color white.*

As shown in Figure 8, there are two places traffic becomes "color white" in Jupiter – **a)** after passing the spine block and bouncing back to $S_{4b}$ switches and **b)** after reaching the destination server block's $S_2$ switches. For **a)**, $h_4$ is applied on $S_{4a}$ towards the $S_{5a}$ and $S_{5b}$ direction. $S_{5a}$ and $S_{5b}$ apply an orthogonal hash function $h_5$, and therefore, the same portion (50%) of $S_{4a} \rightarrow S_{5a}$ and $S_{4a} \rightarrow S_{5b}$ goes to $S_{4b}$. Effectively, the traffic of $S_{4a} \rightarrow S_{5a}$ and $S_{4a} \rightarrow S_{5b}$ towards $S_{4b}$ recombines and $h_4$ can be reused. We define "polarized traffic w.r.t $h_4$" as the unequally bucketized traffic through $S_{4a} \rightarrow S_{5a}$ and $S_{4a} \rightarrow S_{5b}$ respectively due to $h_4$. For **b)**, traffic gets polarized w.r.t $h_2$ on $S_2$, however, after traffic reaching destination server block's $S_2$ chips, each $S_2$ is designed to receive the same portion of polarized traffic w.r.t $h_2$, therefore $h2$ can be reused on $S_2$ for downward traffic hashing. The sufficient condition of color recombining is *there exists a "color recombining stage" such that each switch in this stage receives the same portion of polarized traffic w.r.t $H_x$ and after this color recombining stage, $H_x$ can be reused without incurring hash correlation.* With color recombining, we effectively reduce the number of independent hash functions needed in Jupiter from 8 to 6 and all of the switch chips we use can meet this number.

Per-port hashing with color recombining inherits the pros of per-port hashing while requiring less hash functions from commodity switches. However, we found that this scheme breaks for the emerging spineless DCN topology [32]. In spineless DCN, both direct paths and transit paths are employed to route traffic because network architects want to a) increase path diversity to improve availability and b) per-

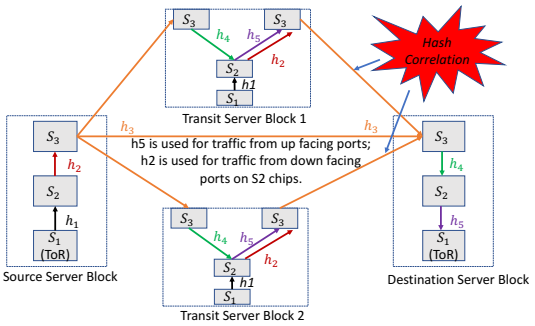



Figure 10: Per-port hashing breaks ($h_3$ appears twice on the forwarding path) in spineless DCN with non-shortest path routing.

form traffic engineering to hedge against unpredictable traffic spikes. Figure 9 shows that if we restrict routing to direct paths only, the per-port hashing scheme still works. However, as shown in Figure 10, we need to use $h_3$ twice for the flows traversing through a transit server block where the packet forwarding path is $S_1(source\,ToR) \rightarrow S_2 \rightarrow S_3 \rightarrow S_3 \rightarrow S_2 \rightarrow S_3 \rightarrow S_3 \rightarrow S_2 \rightarrow S_1(destination\,ToR)$. Please note that traffic traversing through the transit server block bounces through $S_2$ chips for load balancing purposes. As mentioned in Section 2, we need $O(N)$ independent hash functions where $N$ is the number of server blocks. $N$ is a large number in our fabrics, so we need to identify more generic techniques to mitigate hash correlation. In Section 4, we will discuss the *coprime* technique for mesh topologies such as spineless DCN or WAN.

## 4 Mitigating Correlation for Mesh Networks

In this section, we describe a generic approach based on *coprime theory* to mitigate hash correlations in mesh networks. We first provide a theory about coprime in §4.1. Following it, we describe how the coprime theorem can be used to mitigate hash correlations for both ECMP and WCMP, in §4.2 and §4.3, respectively.

### 4.1 The Coprime Theorem

The key idea of the coprime theory is the modulo operation on coprime numbers (e.g., 127 and 128 are two coprime numbers) makes a hash function's output uncorrelated. Considering one hash function $H$ hashed to $\{0,1,...,\hat{H}\}$, we propose to apply the modulo operation of two coprime values to derive two independent hash functions $H_1$ and $H_2$ from $H$, where $\hat{H}$ is the highest hash value.

Below we explain how we use the coprime theory to mitigate hash correlation between two switches. In a switch, we use a hash function to choose the next hop via performing a modulo operation, i.e., $H(x)\%m$, where $H(x)$ is a hash value on a packet $x$, and $m$ is the number of next hops in the ECMP or WCMP group. Considering a scenario where two switches on a forwarding path both use $H$ to choose the next hop, there exists hash correlation and traffic polarization as described in Figure 4b. Instead of using the same hash function $H$ in these two switches, as denoted in Equation 1, we can use the derived $H_1$ ($H_1 = H\%q_1$) on the first switch to choose a next hop among $m_1$ next hops and $H_2$ ($H_2 = H\%q_2$) on the second switch, hashed to $m_2$ next hops. $q_1$ and $q_2$ are coprime numbers. The theorem 2 shows the two hash functions have no correlations.

$$H_i = H\%q_i, i \in \{1,2\} \tag{1}$$

where $q_1$ and $q_2$ are two coprime values, and $q_i < \hat{H}$.

**Theorem 2** $\forall i, \forall j, \Pr(H_2(x)\%m_2 = j | H_1(x)\%m_1 = i) \simeq \Pr(H_2(x)\%m_2 = j)$ *if the following two conditions are satisfied:*

*Condition 1: $q_1 \gg m_1$ or $q_1\%m_1 = 0$, and $q_2 \gg m_2$ or $q_2\%m_2 = 0$;*

*Condition 2: $\hat{H} \gg q_1 q_2$.*

*where $q_1$ and $q_2$ are two coprime values, $m_1$ and $m_2$ are the number of next hops, and $x$ is a packet.*

The theorem shows that the hash value of $H_2\%m_2$ is independent with the hash value of $H_1\%m_1$ for an input $x$ when choosing proper coprimes $q_1$ and $q_2$ ($q_1$ and $q_2$ should be chosen to meet condition 1 and 2 of Theorem 2). The proof of Theorem 2 can be found in the appendix.

### 4.2 Coprime for ECMP

Based on Theorem 2, we propose a coprime-based approach to mitigate hash correlations along a routing path. When two hash functions are correlated, we just choose two coprimes and apply an extra modulo operation to derive two independent hash functions as in Equation 1.

While it may seem to be intuitive and easy to add an extra modulo operation to the hash value in a switch, but this requires switch hardware modification and no switch chips we are using provide this functionality. Even if switch vendors provide this functionality in their next generation chips, we have to replace all our existing switches, which is daunting and costly.

Instead, we propose to duplicate the ECMP group entries to match the coprime value in the SDN controller and interact with the switch flow and group tables via the existing OpenFlow [21] interface. Figure 11 shows the procedure of the method. We explain this using an example. Assuming that there are two egress ports in the ECMP group for IP prefix 10.1.2.0/24. Suppose one hash correlation occurs, we choose a coprime value of 5 to mitigate the hash correlation. Instead of modifying switch hardware to achieve two modulo operations $h\%5\%2$ ($h$ is a hash value returned from the hash function) to choose an egress port, we duplicate 2 physical

egress ports into 5 logical ports in the ECMP group which are mapped to 2 physical egress ports. In this way, we only need one modulo operation, that is, $h\%5$, to determine the logical port and finally the physical egress port. Duplicating ECMP group entries to match the coprime value is supported by today's commodity switches and we only need to add a small amount of logic into the SDN controller. Note that Figure 11 shows that one flow table contains many IP prefixes/flows sharing the same group (multi-path) table in the switch, so we need to use an offset added to *hash value%group size* to index each IP prefix/flow.

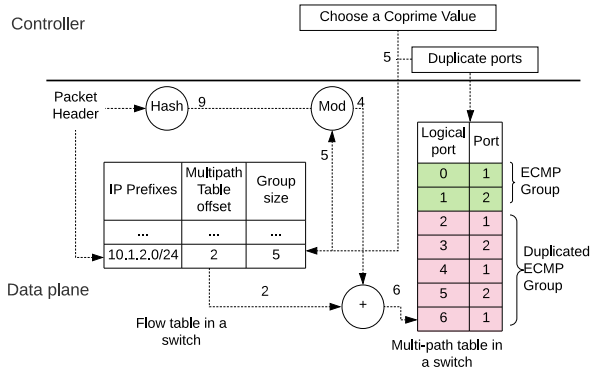
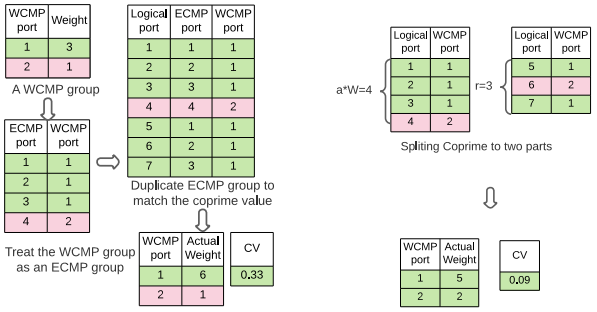



Figure 11: The procedure of applying a coprime number to mitigate hash correlation without requiring an extra modulo operation.

There are two technical challenges of the *copriming ECMP group size* method mentioned above: 1) increased switch memory usage, especially for large coprime values. The memory usage is increased by $O(q/m)$ times, where $q$ and $m$ is the coprime and original ECMP group size, respectively; and 2) ECMP precision loss – i.e., the difference between intended weights and actual weights of different egress ports of an ECMP group. In order to save switch memory, we may want to choose small coprime values. However, small coprime value contradicts condition 1 in Theorem 2; furthermore, we also need to tolerate the ECMP precision loss introduced by a small coprime value. When copriming ECMP group size, some ports are duplicated for $\lfloor q/m \rfloor$ times, and others are duplicated for $\lfloor q/m \rfloor + 1$ times. So when $q/m$ is small, this introduces ECMP precision loss.

We use the coefficient of variation (CV) to measure how effective a coprime value is. Suppose we have $m$ links, and the expected portion of traffic is $p_i = 1/m$ per link. After using coprime $q$, the actual traffic distribution is $\hat{p}_i = (\lfloor q/m \rfloor + I(i \leq q\%m))/q$, where $I(.)$ is an indicator function, $i \in \{0, m-1\}$ is the port id, and when $i \leq q\%m$, $I(.) = 1$. The CV is computed for $\hat{p}_i$. In our implementation of the coprime method, we minimize CV while obeying the ECMP table size limit offered by switch chips.




(a) Naive coprime: the WCMP group is treated as an ECMP group with 4 ECMP ports.

(b) Split coprime: The coprime value 7 is split into the first $\hat{q} = a*W = 4$ and the remaining $r = 3$, where $a = \lfloor q/W \rfloor = 1$.

Figure 12: Illustration and comparison of the naive and improved algorithm to coprime WCMP groups.

## 4.3 Coprime for WCMP

As discussed in [37], topology asymmetry introduced by link or switch failures requires Weighted-Cost Multi-Path (WCMP) to distribute traffic in proportion to downstream hops' capacities. In this section, we extend the coprime-based method from ECMP to WCMP.

One straightforward method, denoted by *naive coprime*, is to treat a WCMP group as an ECMP group of $W$ ECMP ports, where $W$ is the sum of the weights, i.e., $W = \sum_i w_i$ and $w_i$ is the weight of port $i$. We duplicate the $W$ ECMP ports to $q$ logical ports just as an ECMP group, where $q$ is a coprime value. However, the weights after duplication could deviate significantly from the intended WCMP weights. One example is shown in Figure 12a: there are two ports in the WCMP group, and their weights are 3 and 1, that is, $w_1 = 3$ and $w_2 = 1$. This WCMP group can be treated as an ECMP group with $W = 4$ ECMP ports. Suppose, we choose a coprime value 7, and after duplication, the actual weights are $\hat{w_1} = 6$ and $\hat{w_2} = 1$, which are significantly different from the intended WCMP weights. We use the CV of $\{\hat{w_i}/w_i\}$ to quantify the difference. As shown in Figure 12a, the CV is 0.33.

To reduce the difference between the actual weights after copriming WCMP group size and the expected WCMP weights, we propose an improved algorithm to duplicate entries in the WCMP group, denoted by *split coprime*. Suppose the WCMP group has $m$ ports, the weight is $w_i$ for port $i$, $W = \sum_i w_i$, and the chosen coprime value is $q$. We split the coprime value to two parts: $\hat{q} = a*W$ and $r = q\%W$, where $a = \lfloor q/W \rfloor$. It is intuitive to duplicate the ports to $\hat{q}$ logical ports, that is, each WCMP port are duplicated for exactly $w_i * a$ times. We duplicate $m$ ports to left $r$ entries in the following manner: each port $i$ is replicated for $\lfloor r/m \rfloor + I(i < r\%m)$ times, where $I(.)$ is an indicator function, and when $i < r\%m, i \in \{0, 1, ..., m-1\}$, $I(.) = 1$. Figure 12b illustrates the procedure of co-priming an WCMP group: the

coprime value $q = 7$ is split into $\hat{q} = 4$ and $r = 3$; for $\hat{q}$, the two WCMP ports are duplicated for $w_i$ times, and $w_1 = 3$ and $w_2 = 1$; For $r$, port 1 is replicated for two times and port 1 for once. With this improved algorithm, the CV is much smaller than the naive way of simply treating a WCMP group as an ECMP group.

For WCMP, the memory cost of coprime is negligible because even without coprime, we need to do WCMP quantization (i.e., approximating fractional weights via duplicating ECMP table entries) [37] to ensure the number of WCMP entries in a group doesn't exceed a pre-defined limit.

## 5 Evaluation

We conducted simulations using three types of network topologies and real-world traffic traces. We also evaluated on a hardware testbed with hundreds of switches and large-scale production fabrics.

### 5.1 Experiment Setup

**Traffic traces** We use CAIDA trace dataset [5] from a high-speed Chicago monitor on a commercial backbone link: each 1-second segment has $475.37k$ packets and $12.91k$ 5-tuple flows on average. Hashing is the foundation of traffic load balancing in multi-path networks and it is applied at the flow-level in ECMP/WCMP routing, therefore we focus on flow-level statistics rather than packet-level in our simulation to quantify the goodness of a hashing design. To map a traffic trace to a network topology we evaluate, we randomly assign each IP in the traffic trace to one host in our topology.

**Network topologies** We employ three types of topologies in our evaluation (Table 1): 1) the simplified multi-stage Clos (Jupiter) DCN topology as depicted in Figure 1; 2) a spineless DCN which has eight server blocks and the connection among server blocks is DRing [13]. Each server block contains 8 $S_3$ chips and 8 $S_2$ chips (each chip has 16 ports). Every server block is directly connected to 4 neighboring server blocks and each server block pair has 16 direct links; 3) two ISP topologies [27]. The routing strategy for the ISPs is $k$-shortest ($k = 4$) path routing [31] in our simulation.

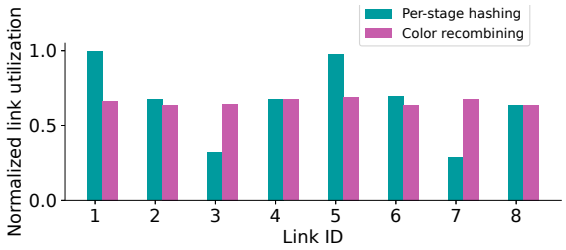



Figure 13: The normalized link utilization of eight links from one ECMP group for color recombining and per-stage hashing.

| Topology | #Nodes | #Links |
|---|---|---|
| Spinefull DCN (Figure 1) | 2 server blocks | 128 |
| Spineless DCN (DRing [13]) | 8 server blocks | 512 |
| ISP1 (small) | 69 | 146 |
| ISP2 (large) | 122 | 371 |

Table 1: The topologies used in our simulation.

**Hash functions** We use one of the most widely used hash function family, RTAG7 [10], which includes seven hash functions (6 CRC16s and 1 CRC32) in our simulation.

**Metrics** We employ the Coefficient of Variation (CV), which is defined as $\delta/\mu$ ($\delta$ is the standard deviation and $\mu$ is the mean of a set of data points), to quantify the goodness of hashing. CV is a commonly used statistical measure of the dispersion of data points around the mean. For an ECMP group with $m$ output ports and the associated link utilization set $\{x_i\}$, $1 \le i \le m$, CV is computed against set $\{x_i\}$ directly. For a WCMP group with $m$ output ports whose weighs are $w_i$, $1 \le i \le m$, CV is computed against set $\{x_i/w_i\}$.

### 5.2 Color Recombining for Multi-stage DCN

We first study the traffic load-balancing performance of the proposed per-port hashing with color recombining (denoted as *color recombining* below) scheme for multi-stage DCN and compare with per-stage hashing with random seed (denoted as *per-stage hashing*). For per-stage hashing, we assign an independent hash function for each stage and initialize each hash function with a random seed.

We present normalized link utilization of eight links from one ECMP group in Figure 13. It shows that all eight links have similar link utilization of around 0.67 for color recombining, but the per-stage hashing approach shows severe non-uniformity. In color recombining, we only reuse hash functions when color recombining happens, which eliminates hash correlation; but in per-stage hashing, certain hash functions are reused without considering correlations and random seeds are linear operations that can not decorrelate the reused hash functions.

We also compute the CV for each ECMP group and draw the CDF of CVs in Figure 14. All the CVs are under 0.05 for color recombining, but per-stage hashing's CV can be above 0.6 (note that a CV of 1 means the standard deviation is equal to the mean). In other words, per-port hashing with the color recombining approach reduces CV by approximately an order of magnitude compared with per-stage hashing with random seeds. Large CV value means links in the same ECMP group are not properly utilized and results in wasted network capacity and unnecessary hot links that lead to network congestion under heavy traffic load.

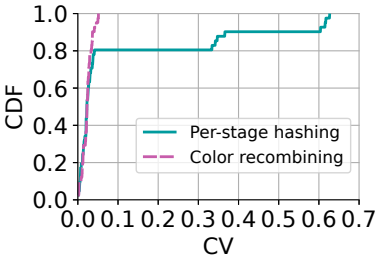



Figure 14: The CDF plot of CVs of each ECMP group. Compare color recombining with per-stage hashing.

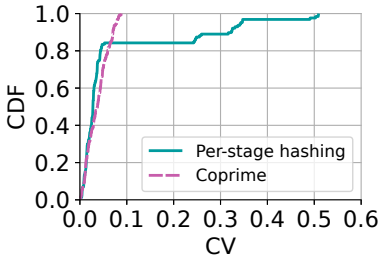



Figure 15: The CDF plot of CVs of each ECMP group. Compare coprime with per-stage hashing.

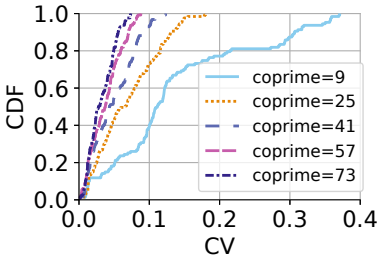



Figure 16: The CDF plot of CVs of each ECMP group. Compare different coprime values.

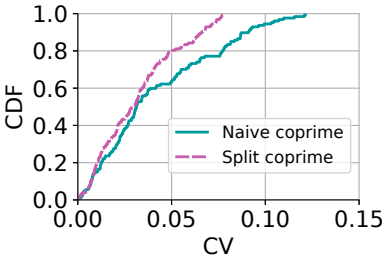



Figure 17: The CDF plot of CVs for each WCMP group. Compare two algorithms of copriming WCMP group.

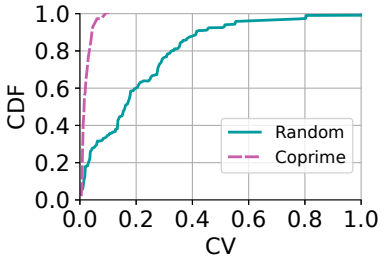



Figure 18: CDF plot of CV of each ECMP group for ISP 1.

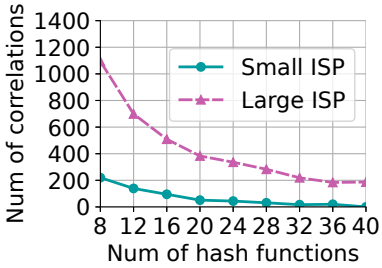



Figure 19: The number of ECMP groups of $CV > 0.1$.

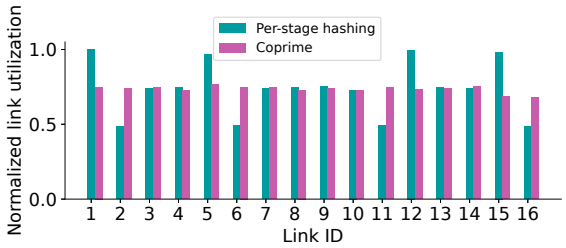



Figure 20: The normalized link utilization of 16 links connecting two server blocks.

## 5.3 Coprime for Spineless DCN

In the spineless DCN, all eight server blocks are connected in a DRing topology [13]. We choose two coprime values for every server block pair to mitigate the hash correlation between them. We compare coprime with the per-stage hashing where each server block uses 3 randomly chosen hash functions from the RTAG7 hash family and each function is supplied a random seed. We conduct experiments to evaluate the coprime-based approach for both ECMP and WCMP.

### 5.3.1 Coprime for ECMP

We show the normalized link utilization from 16 links connecting two server blocks in Figure 20. All links have the utilization of around 0.75 for the coprime-based approach, where the coprime values are 8 (note each ECMP group has 8 ports because each $S_3$ chip of a server block has 8 up-facing ports) and 57 for the two server blocks, respectively. However, using per-stage hashing, the max/min link utilization ratio is above 2. Due to the shortage of independent hash functions, per-stage hashing has to reuse certain identical hash functions (even though they are provided with random seeds), and this raises traffic polarization as shown in Figure 20.

For quantification, the coprime-based approach outperforms per-stage hashing by reducing the CV by about 80%. For the coprime-based approach, all CVs are under 0.1, but for per-stage hashing, the CV can be as high as 0.5, as shown in Figure 15.

The coprime value matters when mitigating hash correlation: a large coprime value is more effective than a small one. We evaluate five different coprime values from 9 to 73, and show the CVs in Figure 16. It shows that the CV can be close to 0.4 when the coprime value is 9. When increasing the coprime value to 73, the improvement is not significant compared to 57. The result is consistent with our analysis in the end of § 4.2, which describes a trade-off between memory usage and ECMP precision. How to choose a coprime value depends on the memory a switch has and the level of imbalance one can tolerate.

### 5.3.2 Coprime for WCMP

To evaluate coprime for WCMP, we simulate a scenario where the first half links in an ECMP group have 2× capacity, and therefore, the weights for those links are 2 and the remaining ones are 1. In this evaluation, we focus on two ways to duplicate the WCMP ports to match the coprime value: one is the *naive coprime* where a WCMP group is regarded as an ECMP as shown in Figure 12a; the other one (denoted as *split coprime*) is to split the coprime value into two parts as described in Figure 12b.

We draw the CDF plot of CVs per WCMP group in Figure 17. It shows that all CVs are under 0.075 when using the split coprime algorithm, while CV can reach 0.12 when using the naive coprime algorithm. We also observe split coprime reduces CV by approximately 60% for some WCMP groups compared with naive coprime. Our results indicate that split coprime should be preferred over naive coprime because it reduces CV using the same amount of switch memory.

## 5.4 Coprime for WAN

The coprime-based approach works for WAN topologies. We use the two ISP networks in Table 1 to evaluate load balancing performance of the coprime technique. We compare the coprime method with a random hash allocation approach (denoted as *Random* in Figure 18), where each switch randomly chooses an independent hash function from RTAG7 hash family and applies a random seed.

Our result shows that the coprime-based approach reduces CVs by about one order of magnitude compared with the random method. We compute all CVs for the link utilizations in every ECMP group and draw the CDF plot of CVs in Figure 18. When using the random method, CV can be as large as 1, which means only one link in the ECMP group is used, and others are idle. On the other hand, applying a coprime value to the ECMP group increases load balancing performance significantly, with CV lower than 0.1.

We also study how many independent hash functions are enough to eliminate traffic imbalance. We conduct an experiment by increasing the number of hash functions and gather the number of ECMP groups of $CV > 0.1$, caused by hash correlations. For this experiment, we introduce more CRC hash functions [24] beyond RTAG7, without considering the hardware implementation limitation. The result is shown in Figure 19. It shows that the small ISP (ISP 1) requires at least 40 hash functions to eliminate the correlations. However, the larger ISP (ISP 2) still has nearly 200 correlations when using 40 hashes. This result implies that even if chip vendors provide more hash functions in their next generation chips, it might be not sufficient for a large topology, e.g., B4 WAN grew 7× larger in 5 years [15]. On the other hand, the coprime-base algorithm can be commonly used for topologies of arbitrary sizes.

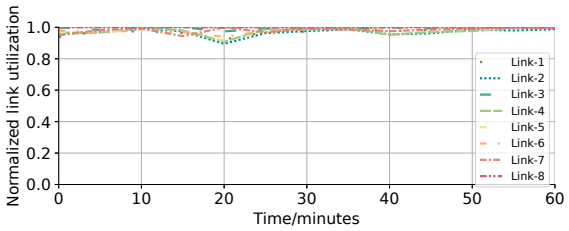



Figure 21: The normalized link utilization of one $S_3$ switch over 1-hour time window from the hardware testbed.

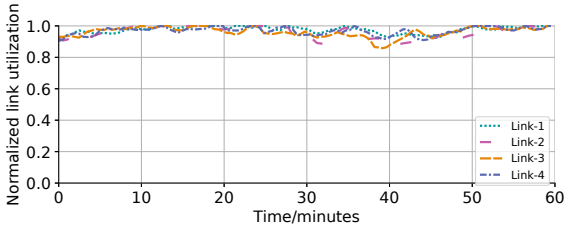



Figure 22: The normalized link utilization of four links from one spine block towards one server block over 1-hour time window from the production fabric after applying color recombining.

## 5.5 Hardware Testbed Evaluation and Production Fabric Deployment

We constructed a hardware testbed using the spineless DCN topology (Figure 3). There are 4 server blocks and hundreds of switches in this hardware testbed. The testbed serves a few Tbps traffic generated by production-grade applications and is carried by TCP and UDP. Within each server block, we apply a per-port hashing scheme as Figure 9 shows and in total 5 independent hash functions are used. We allow both shortest path routing and non-shortest path routing, i.e., we allow traffic to transit through an intermediate server block to reach a destination server block. To mitigate the hash correlation problem as illustrated in Figure 10, we apply the coprime scheme to coprime ECMP group size on $S_3$ switches – for traffic originating a server block, ECMP group size is coprime to 128; for traffic transiting a server block, ECMP group size is coprime to 127. Therefore, we can ensure there is no hash correlation for both shortest and non-shortest routing paths. We show the normalized link utilization of one $S_3$ switch over 1-hour time window in Figure 21. We observe excellent load balancing performance with a CV of 0.02.

Color-recombining has been deployed in our production multi-stage Clos fabrics for several years and significantly reduced load imbalance due to hash correlation. The deployment of color-recombining is the following: after deciding which hash functions can be reused, we simply configure the switches with the designated hash functions. We studied the hashing performance result of color-recombining in pro-

duction multi-stage Clos fabrics and the normalized link utilization of one representative spine block towards one server block from $S_4$ to $S_3$ direction is shown in Figure 22. The spine block's port speed is 40Gbps. As we discussed in Section 3 (Figure 8), hash function $h_4$ is reused to load balance traffic leaving spine blocks from $S_4$ to $S_3$ direction. But due to the color-recombining technique, polarized traffic due to $h_4$ are merged into color white when leaving $S_4$ chips, as a result, we observe very nice load balancing performance with a CV of 0.03.

## 6 Related Work

**Traffic Load Balancing** There are many prior works to address an important limitation of ECMP/WCMP: load balancing performance degrades when there is a large traffic entropy, i.e., when elephant flows collide on the same path, network congestion arises. For example, previous works [3, 28] propose to split elephant flows into smaller "flowlets" that can be load-balanced over different paths; other works [2, 30] propose to reschedule elephant flows after detecting collisions. MPTCP [29] is a transport protocol that uses subflows to transmit over multiple paths. These works rely on the assumption that there is no hash correlation. Our work is complementary and all load balancing schemes benefit from our improved hashing design, which is the corner stone of load balancing.

**Hashing in Networks** The universal algorithm [7] adds a 32-bit router-specific value to the hash function; however, as we show theoretically and experimentally in §2.3, random seeds do not work well. The paper [36] proposes to mitigate hash function correlation by randomly setting the VLAN-id for each switch. However, randomizing the VLAN-id increases network management complexity. The paper [14] selects a different hash function for a different value of the TTL field. However, this approach is still constrained by the limited number of hash functions and it also requires modifying switch hardware. The novelty of this paper is that our approaches work with commodity switch hardware without any hardware modification or switch upgrade, which is costly and daunting in large-scale networks.

**Decorrelate Hashing** Researches have already relied on prime numbers to design independent hash functions. For example, the universal hash functions employs the prime number as the divisor [6, 23], where primes are special case of coprimes. Our work extends prior theory and applies it to improve traffic load balancing in modern networks.

The patent by A. Meyer [22] uses co-primes to solve the storage collision for hash tables, that is, how to insert an item into the hash table when collision occurs; however, their method is different from ours where they add a co-prime offset to the original hash output, and this is similar to choosing an initial value for the hash function. We have proved that random initial values (including co-prime ones) do not work for an ECMP group of even size in Theorem 1.

## 7 Conclusions

This paper tackles a real but underestimated problem in network traffic load balancing, i.e., traffic polarization caused by hash correlations. This paper proposes two novel approaches to mitigate hash correlation: 1) a color recombining technique which exploits topology traits of Clos networks to allow hash function reuse and to reduce the number of needed independent hash functions in multi-stage Clos DCN and 2) a generic coprime-based technique to mitigate hash correlation for non-hierarchical mesh networks such as spineless DCN and WAN. Evaluations results based on real traffic trace and topologies show that the proposed techniques can reduce the extent of load imbalance, quantified by coefficient of variance (CV), by one order of magnitude. The limited hashing capability offered by current switch silicon reduces network reliability, efficiency and poses challenges to modern large-scale networks. We believe that novel approaches that work with current switch hardware are valuable. We also hope our use cases of more flexible topology and routing designs can motivate switch vendors to provide better hashing support in the future.

## 8 Acknowledgement

We thank our anonymous shepherd and reviewers for their helpful suggestions. The authors thank David Wetherall, Abdul Kabbani and Christophe Diot for their insightful feedback which greatly improves the quality of this paper. This work was supported in part by the National Science Foundation (grants CNS-1618030 and CNS-2107078).

## References

[1] Venkata Ramana Kiran Addanki. Method and system for management of flood traffic over multiple 0: N link aggregation groups, April 22 2014. US Patent 8,705,551.

[2] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–19, 2010.

[3] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Francis Matus, Rong Pan, Navindra Yadav, George Varghese, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 503–514. ACM, 2014.

[4] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center

tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[5] CAIDA. The CAIDA anonymized internet traces data access. http://www.caida.org/data/passive/passive_dataset_download.xml, 2019.

[6] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. *Journal of computer and system sciences*, 18(2):143–154, 1979.

[7] Cisco. Cef polarization. https://www.cisco.com/c/en/us/support/docs/ip/express-forwarding-cef/116376-technote-cef-00.html, 2013.

[8] Cisco. Data center access design with cisco nexus 5000 series switches and 2000 series fabric extenders and virtual portchannels. https://itnetworkingpros.files.wordpress.com/2014/04/c07-572829-01_design_n5k_n2k_vpc_dg.pdf, 2018.

[9] Broadcom Corporation. Bcm56070 switch programming guide. https://docs.broadcom.com/doc/56070-PG2-PUB, 2020.

[10] Dell. Dell configuration guide for the s4048-on system 9.9(0.0). https://www.dell.com/support/manuals/us/en/19/force10-s4048-on/s4048_on_9.9.0.0_config_pub-v1/rtag7, 2015.

[11] Dell. Dell networking configuration guide for the mxl 10/40gbe switch i/o module 9.9(0.0). http://www.dell.com/support/manuals, 2015.

[12] Network Working Group. Ip in ip tunneling. https://datatracker.ietf.org/doc/html/rfc1853.

[13] Vipul Harsh, Sangeetha Abdu Jyothi, and P Brighten Godfrey. Spineless data centers. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 67–73, 2020.

[14] Ariel Hendel. Mutable hash for network hash polarization, March 19 2015. US Patent App. 14/026,725.

[15] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, et al. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 74–87, 2018.

[16] C. Hopps. Analysis of an equal-cost multi-path algorithm. RFC 2992, RFC Editor, November 2000.

[17] Yuanhong Huo, Xiaoyang Li, Wei Wang, and Dake Liu. High performance table-based architecture for parallel crc calculation. In *The 21st IEEE International Workshop on Local and Metropolitan Area Networks*, pages 1–6. IEEE, 2015.

[18] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[19] Abhijeet Joglekar, Michael E Kounavis, and Frank L Berry. A scalable and high performance software iscsi implementation. In *FAST*, volume 5, pages 267–280, 2005.

[20] Nick McKeown. Software-defined networking. *INFOCOM keynote talk*, 17(2):30–32, 2009.

[21] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.

[22] Alex Meyer. Co-prime hashing, July 28 2020. US Patent 10,725,990.

[23] Mats Näslund. Universal hash functions & hard core bits. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 356–366. Springer, 1995.

[24] Carnegie Mellon University Philip Koopman. Best crc polynomials. https://users.ece.cmu.edu/ koopman/crc/.

[25] R. Wang, H. Wassel, J. Zhou, B. Felderman and D. Wetherall. Experiences with multipath forwarding in dc networks. 2017 Google Networking Research Summit.

[26] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *ACM SIGCOMM computer communication review*, volume 45, pages 183–197. ACM, 2015.

[27] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 133–145. ACM, 2002.

[28] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *NSDI*, pages 407–420, 2017.

[29] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath tcp. In *NSDI*, volume 11, pages 8–8, 2011.

[30] Xin Wu, Daniel Turner, Chao-Chih Chen, David A Maltz, Xiaowei Yang, Lihua Yuan, and Ming Zhang. Netpilot: automating datacenter network failure mitigation. *ACM SIGCOMM Computer Communication Review*, 42(4):419–430, 2012.

[31] Jin Y Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27(4):526–530, 1970.

[32] Mingyang Zhang, Jianan Zhang, Rui Wang, Ramesh Govindan, Jeffrey C Mogul, and Amin Vahdat. Gemini: Practical reconfigurable datacenter networks with topology and traffic engineering. *arXiv preprint arXiv:2110.08374*, 2021.

[33] Zhehui Zhang, Haiyang Zheng, Jiayao Hu, Xiangning Yu, Chenchen Qi, Xuemei Shi, and Guohui Wang. Hashing linearity enables relative path control in data centers. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 855–862, 2021.

[34] Rui Zhang-Shen and Nick McKeown. Designing a predictable internet backbone with valiant load-balancing. *Quality of Service–IWQoS 2005*, pages 178–192, 2005.

[35] Shizhen Zhao, Rui Wang, Junlan Zhou, Joon Ong, Jeffrey C Mogul, and Amin Vahdat. Minimal rewiring: Efficient live expansion for clos data center networks. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 221–234, 2019.

[36] Junlan Zhou and Zhengrong Ji. Hashing technique to optimally balance load within switching networks. 2017.

[37] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. Wcmp: Weighted cost multipathing for improved fairness in data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, page 5. ACM, 2014.

## A Proofs of Theorems

**Proof of Theorem 1 (CRC Seed)** The CRC seed is also known as the initial value, where we can initialize the CRC value via XORing the seed with the input byte. Suppose the input is denoted by $x$, and the two random seeds are $z_1$ and $z_2$. In order to prove random seeds are not effective, we only need to prove the following two rules:

If $crc_b(x)\%2 = crc_b(y)\%2$, $crc_b(x \oplus z_1)\%2 = crc_b(y \oplus z_1)\%2$;

If $crc_b(x)\%2 = crc_b(y)\%2$, $crc_b(x \oplus z_2)\%2 = crc_b(y \oplus z_2)\%2$.

Let $z_1 = crc_b(t)$. Based on the rolling property of the CRC function, we get:

$crc_b(x \oplus z_1) = crc_b((t \ll b_x)|x) = crc_b(t \ll b_x) \oplus crc_b(x)$.

where $b_x$ is the binary length of $x$, $\oplus$ is XOR and $|$ is bitwise and. We also have

$crc_b(y \oplus z_1) = crc_b(t \ll b_x) \oplus crc_b(y)$.

If $crc_b(x)$ and $crc_b(y)$ are both even (odd), $crc_b(t \ll b_x) \oplus crc_b(x)$ and $crc_b(t \ll b_x) \oplus crc_b(y)$ are both even or odd, that is, $crc_b(x \oplus z_1)$ and $crc_b(y \oplus z_1)$ are both even(odd).

It is the same for $crc_b(x \oplus z_2)\%2 = crc_b(y \oplus z_2)\%2$

**Proof of Theorem 2 (co-primes)** Let $U, W, U'$ and $W'$ denote the distribution of the hashing output of $H\%m_1$, $H\%m_2$, $H\%q_1\%m_1$ and $H\%q_2\%m_2$, respectively. When $\hat{H} \gg q_1q_2$ (condition 2 in Theorem 2), $\forall i' \in [0, q_1-1]$, $j' \in [0, q_2-1]$, $i \in [0, m_1-1]$, and $j \in [0, m_2-1]$, we have,

$$\Pr(U' = i', W' = j') = 1/(q_1q_2)$$

When condition 1 is not satisfied in Theorem 2, there exists hash correlation brought by $q_1\%m_1$ and $q_2\%m_2$ because the mapping from $q_1(q_2)$ to $m_1(m_2)$ has a remainder. This mapping causes the non-uniform distribution of $q_1$ integers over $m_1$ slots, where $q_1\%m_1$ slots get $\lfloor q_1/m_1 \rfloor + 1$ integers each, and $m_1 - q_1\%m_1$ slots get $\lfloor q_1/m_1 \rfloor$ integers each. We denote this non-uniformity by the term *the approximation error*. Note that when $q_1$ increases, the difference between $(\lfloor q_1/m_1 + 1 \rfloor)/q_1$ and $(\lfloor q_1/m_1 \rfloor)/q_1$ can be reduced, and thus, we can quantify the approximation error by Equation (2).

$$\text{err} = (q_1\%m_1)/q_1 + (q_2\%m_2)/q_2 \tag{2}$$

where err denotes the approximation error. When condition 1 is satisfied, err $\approx 0$. Let $S(i) = \{i' | i'\%i = 0\}$, and $S(j) = \{j' | j'\%j = 0\}$, we have,

$$\begin{aligned} \Pr(W = j | U = i) &= \frac{\sum_{i' \in S(i), j' \in S(j)} P(U' = i', W' = j')}{\sum_{i' \in S(i)} P(U' = i')} \\ &= \frac{(q_1/m_1)(q_2/m_2)(1/(q_1q_2))}{(q_1/m_1)(1/q_1)} \\ &= 1/m_2 = \Pr(W = j) \end{aligned} \tag{3}$$