Hashing Linearity Enables Relative Path Control in Data Centers

Zhehui Zhang, *University of California, Los Angeles;* Haiyang Zheng, Jiayao Hu, Xiangning Yu, Chenchen Qi, Xuemei Shi, and Guohui Wang, *Alibaba Group*

https://www.usenix.org/conference/atc21/presentation/zhang-zhehui

This paper is included in the Proceedings of the 2021 USENIX Annual Technical Conference.

July 14–16, 2021

978-1-939133-23-6

Open access to the Proceedings of the 2021 USENIX Annual Technical Conference is sponsored by USENIX.

Hashing Linearity Enables Relative Path Control in Data Centers

Zhehui Zhang[1], Haiyang Zheng[2], Jiayao Hu[2], Xiangning Yu[2], Chenchen Qi[2], Xuemei Shi[2], Guohui Wang[2]
[1]*University of California, Los Angeles,* [2]*Alibaba Group*

Abstract

A data center network is an environment with rich path diversity, where a large number of paths are available between end-host pairs across multiple tiers of switches. Traffic is split among these paths using ECMP (Equal-Cost Multi-Path routing) for load balancing and failure handling. Although it has been well studied that ECMP has its limitations in traffic polarization and path ambiguity, it remains the most popular multi-path routing mechanism in data centers because it is stateless, simple, and easy to implement in switch ASICs.

In this paper, we analyze the ECMP hash algorithms used in today's data center switch ASICs, aiming for lightweight path control solutions that can address the ECMP limitations without any changes to existing data center routing and transport protocols. Contrary to common perceptions about the randomness of ECMP hashing, we reveal the linear property in the hash algorithms (e.g. XOR and CRC) used in widely deployed switch ASICs in data centers. Based on the hashing linearity, we propose relative path control (`RePaC`), a new lightweight, and easy-to-deploy path control mechanism that can perform on-demand flow migration with deterministic path offsets. We use a few case studies to show that `RePaC` can be used to achieve orders of magnitude faster failover and better path planning with up to 3 times link utilization gain in hyper-scale data centers.

1 Introduction

To support the high bandwidth and high availability requirements of cloud and big data applications, data center networks are often designed with rich path diversity. Typical data center network topologies, such as FatTree [1], Clos [10], or HyperCube [16], offer hundreds of paths across multiple tiers of switches between any pair of servers. In such a multi-path network, managing a large number of paths among all end-point pairs is challenging. The path selection mechanism is always a key part of data center network design to fully utilize the high bandwidth from these paths and achieve good traffic load balance and fault-tolerance properties.

The multi-path IP routing protocol (e.g. BGP [27]) with ECMP [20] is the most common routing scheme in many production data centers [13, 26]. In such networks, reachable network prefixes and available paths are propagated to all the switches using BGP, and each switch uses ECMP to select a next hop based on the hash value from specified packet header fields. To understand the performance of ECMP-based routing, much research has been done. A common perception is that ECMP offers reasonable load balancing and fault tolerance among a large number of uniform flows, however, it is difficult to perform any explicit path control on ECMP because it is stateless, and its hashing calculation is random [18, 21, 46]. Besides, ECMP suffers from traffic polarization because it performs static load balancing based on header fields without considering flow sizes [2].

Many alternative path control mechanisms, such as Hedera [2], XPath [21] and Multi-path TCP [36, 37], have proposed to redesign routing and transport layers to better leverage the path diversity. However, these proposals often require a redesign in either the server network stack or routing protocol of data center switches. As a result, although these proposals offer many advantages over ECMP on dynamic load balancing and precise path control, they have seen very limited deployment in today's production data centers. For the path selection mechanism, ECMP remains the common practice because it is stateless, simple, and easy to implement in switch ASICs.

In this work, we take a different approach to address the ECMP limitations in load balancing and failure handling. Given the wide deployment of BGP with ECMP routing, we look for solutions that require minimal changes to server software stack and data center routing protocols. Instead of treating ECMP as a random path selection black-box, we analyze the ECMP path selection process and investigate typical hash algorithms used in the most popular data center switch ASICs on the market. Our study reveals that most widely-deployed switches use XOR, CRC or their variants for ECMP hashing. These hash algorithms hold a linear property ($ECMP(a) \oplus ECMP(b) = ECMP(a \oplus b) \oplus ECMP(0)$),

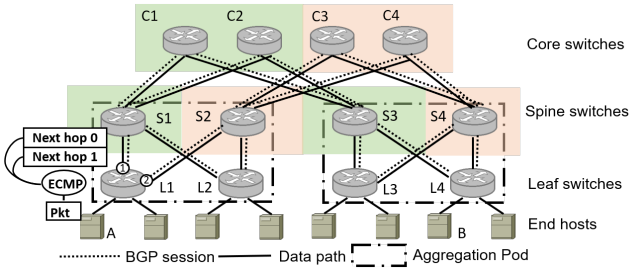


Figure 1: A Clos-based data center network topology

which guarantees a deterministic mapping between packet header changes and path changes. We analyze how hashing linearity affects traffic load balance and how the hashing linearity can be used to control the flow paths to better leverage the path diversity.

We propose a relative path control (`RePaC`) scheme, which uses the deterministic mapping between the header change and path change to perform on-demand flow migration for failure handling and load balancing. We validate the hashing linearity and relative path control scheme on a testbed with the same ECMP configuration as our production data centers. Our evaluation shows that `RePaC` can achieve fast failover and better traffic engineering for TCP and RDMA applications. With `RePaC`, failover speeds up by orders of magnitude compared with existing in-network failover approaches. `RePaC` also outperforms MPTCP with 2(4) subflows in recovery success rate by 36%(21%). In addition, `RePaC`-based traffic engineering reduces flow completion time by up to 25% and improves link utilization by up to three times. Proven to be simple and effective, `RePaC` can be readily deployed in production data centers with no changes required in the routing and transport protocols.

As far as we know, our work is the first study that investigates the linear property of ECMP hash algorithms and discusses its applications to data center path control. We show the feasibility of leveraging the hash algorithm properties to develop flexible path control algorithms for load balancing and failure handling, which contradicts the common perception about the randomness and ambiguity of ECMP path selection. The resulting solution `RePaC` is lightweight, easy to deploy in production data centers, and easy to integrate with a large spectrum of applications. Our work sheds light on a new perspective of multi-path routing, traffic engineering and application design in data centers.

2 Background and Motivation

2.1 Data Center Networking Primer

Most data center networks adopt some variant of a multi-rooted tree topology. Figure 1 shows an example of a Clos-structured data center network with three tiers of switches, which provides abundant paths between any pair of servers to achieve high aggregated bandwidth and tolerate potential link and device failures. To better utilize the available paths in the network, multi-path routing is the key part of the data center network design, with load balancing and fault tolerance as two primary design goals.

Multi-path BGP with ECMP is the most common multi-path routing design in hyperscale data centers [27]. In this design, switches establish BGP sessions among each other over the connected links and choose next hops based on BGP routing information. Multiple equal-cost next hops are grouped as ECMP groups when routes are installed into the routing table in the switch ASIC. When a packet reaches a switch, a random next hop will be selected in the ECMP group, so flows can be evenly distributed among parallel links for load balancing. As shown in Figure 1, if a packet is being sent to switch L3 via switch L1, the BGP session running on L1 will first decide all equal-cost next hops, S1 and S2. In ECMP, each switch distributes packets based on a hashing value calculated from specific packet header fields. ECMP would select one from S1 and S2. If ECMP distributes flows to S1 and S2 evenly, load balancing is fulfilled. To support failover, the in-network BGP sessions send keepalive messages to detect whether the link is still available. In case of failure, BGP tears down the session of the malfunctioning link, and the corresponding next hop is removed from ECMP. Therefore ECMP is critical for both load balancing and failure handling in data center networks.

2.2 The Limitations of ECMP

Many previous studies have shown that ECMP has several key limitations in practice. First, ECMP fails to leverage the path diversity in a lot of scenarios. Hedera [2] shows ECMP may cause significant bandwidth loss when flows are not evenly distributed. The situation becomes worse when large and long-lived flows co-exist. Second, ECMP distributes traffic using random hash algorithms, which makes it difficult to perform precise path control. A lot of fine-grained flow scheduling and traffic engineering mechanisms cannot be done in data center networks over ECMP [5, 21].

Due to the aforementioned limitations, a common perception of ECMP is that ECMP provides decent load balancing among a large number of flows evenly distributed across a large header space. However, ECMP works under strong assumptions about traffic patterns, and it may fail in many cases with traffic polarization, slow failover, and under-utilized links. And it is difficult to perform deterministic path control because ECMP path selection is random.

Several previous studies have proposed alternative solutions to address the limitations of ECMP from both routing and transport layers. For example, Hedera [2] is proposed to perform dynamic flow scheduling based on a traffic matrix using OpenFlow. XPath [21] proposes to enable explicit path control for data center applications using pre-installed routes. FUSO [8] proposes a multi-path loss recovery mechanism

on MPTCP to enable fast failure recovery in data centers. However, these solutions require major redesigns of the data center routing protocol or server network stacks. And major redesigns of routing and transport protocols have huge impacts on the data center applications and daily network operations. As a result, these solutions still have very limited deployment in today's production data centers.

ECMP still has wide deployment in many production data centers for multi-path routing, however, the implementation of the ECMP hashing mechanism remains a mystery for the research community. There isn't a thorough analysis of the details of the ECMP mechanism on popular data center switch ASICs. Volur [47] is the only study we have seen, which tries to model ECMP as a deterministic mapping from headers to paths. It aggregates all the mappings from all switches and uses model replay to predict ECMP path selection. However, Volur still treats ECMP as a black-box model, and a full network model replay incurs too much overhead for real-time traffic engineering and path control in large data centers.

This dilemma of path control in data centers motivates us to dig into the black-box of ECMP implementation of widely-deployed data center switches and look for lightweight solutions to address the ECMP limitations. Such solutions can be incrementally deployed to a large volume of existing data centers and benefit data center applications without a major redesign of the network. We explore the factors that affect ECMP path selection and reveal an interesting property, linearity of hash algorithms used by most merchandise switch ASICs, which could be used to enable a new relative path control scheme.

3 Demystifying ECMP Path Selection

Our analysis of ECMP path selection is based on the ECMP implementation of popular switch ASICs available on the data center switch market. We look into the publicly available open source repositories and documents about the ECMP implementation on widely deployed switch ASICs [6, 12, 31, 33]. We also investigate the ECMP configurations from the widely-adopted open switch abstraction interface (SAI) [34], which is supported by most switch ASIC vendors, such as Broadcom, Barefoot, CISCO, Mellanox, and Marvell [29]. We consider the variations of ECMP implementations [20, 32, 38] and validate our analysis with commercial switches. Specifically, our validation covers Broadcom Tomahawk series, Trident series, and Barefoot Tofino series [7].

3.1 Modeling ECMP Path Selection

ECMP is typically implemented as part of the routing table matching stage in the switch ASIC pipeline. When multiple routes are available for a given network prefix, equal-cost next hops are added into an ECMP group in the routing table. When a routing table entry is matched for an incoming packet, the packet will be forwarded to one of the next hops in its ECMP group, based on the hashing of the packet header fields [33].

Figure 2 shows a general ECMP processing model from typical switch ASICs. The input of ECMP is header fields, and the output is a next hop ID. There are four stages in ECMP processing: pre-processing, hashing, post-processing, and bucket mapping. We use the following functions to denote the ECMP stages: $Pre\_proc()$, $Hash()$, $Post\_proc()$, $Bucket_B^N()$. The overall ECMP processing can be described as:

$$ECMP(h) = Bucket_B^N(Post\_proc(Hash(Pre\_proc(h)))) \quad (1)$$

where the hash function plays a key role in path selection.

In the $Pre\_proc(h)$ stage, the H-bit input packet header fields h, with range $\{0,1\}^H$, together with a configurable hash seed are processed using bit-wise operations, such as AND, XOR, shifting and masking. For example, if we use source and destination IP addresses and port numbers as input, H would be 96. Then $Hash()$ denotes the hash function from $\{0,1\}^I$ to $\{0,1\}^O$, where I is the length of the pre-processed input and O is the length of the output hash result, both in number of bits. O is usually 32 or 16 since most switches use 32-bit or 16-bit values to represent hash results [12, 31]. In the $Post\_proc()$ stage, the hash result is further shuffled using bit-wise operations. Then $Bucket_B^N()$ performs modulo operations to map the post-processed hash result $\{0,1\}^O$ to one of the next hops in the ECMP group $\{0,...,N-1\}$ placed in B hash buckets, where N is the number of next hops. The entire ECMP process is a mapping from $\{0,1\}^H$ to a number in $\{0, ..., N-1\}$.

3.2 The Linearity of ECMP Process

In this section, we discuss the linear property of the most common ECMP hash functions, and how linearity impacts the path selection.

Hash functions overview: The commercial switch ASICs support various hash functions for ECMP, such as random, XOR, CRC and Pearson hashing [11, 24, 31]. Among them, CRC and XOR are two popular hash algorithms defined in SAI [34], and supported by most switch vendors [29]. There are two reasons that CRC and XOR are widely adopted. These two algorithms have been used in communication systems with mature and efficient ASIC implementation, which includes plenty of circuit optimization [31, 42, 45]. In addition, previous analysis has shown that CRC and XOR algorithms have good load balancing performance given a uniform distribution of flows [49].

The SAI [34] also cites the random hash algorithm, but it is not commonly used in the ECMP implementation of modern switches. The reasons are twofold. First, packet-level random path selection will result in a nightmare in network monitoring

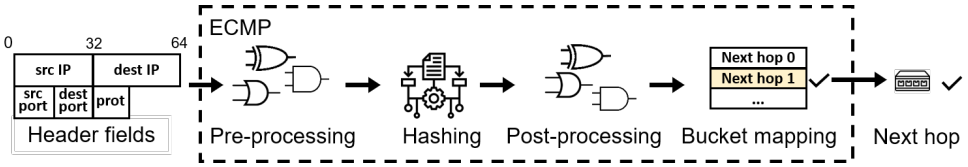


Figure 2: An example of an ECMP packet processing pipeline: pre-process certain header fields, hash and post-process the hash results to get bucket number two, then refer to the bucket for the next hop ID

and troubleshooting. Second, packets from the same flow need to be hashed onto the same path to avoid the out-of-order issues that affect end-to-end transport performance. There are other hash algorithms that are not included in SAI, such as Pearson hashing, which are proprietary of a specific switch ASIC vendor. In this paper, we will focus on the properties of the most common hash functions using CRC and XOR.

The linearity of hash functions: We define the hashing linearity as follows: a hash function is linear, if

$$Hash(h_i) \oplus Hash(h_j) = Hash(h_i \oplus h_j) \oplus Hash(0) \quad (2)$$

where h_i and h_j are arbitrary packet headers, and $Hash(0)$ is the hash result for a packet with all 0's, which is a constant given the initial hashing seed. Both XOR and CRC hashing satisfy hashing linearity. The detailed proof of CRC/XOR hash linearity can be found in supplemental materials [48].

Insight: If the linear hash function is fixed, for any packet header h_i, the hash value after a relative change Δ on the original header fields is deterministic, i.e. $Hash(h_i \oplus \Delta) = Hash(h_i) \oplus Hash(\Delta) \oplus Hash(0)$. In other words, as long as we know the mapping from Δ to $Hash(\Delta)$ and $Hash(0)$, we can predict the relative hash value change.

The linearity of $ECMP()$: We prove the linearity of ECMP by proving the linearity of the other three procedures $Pre\_proc()$, $Post\_proc()$, and $Bucket_B^N()$. The pre-processing and post-processing functions are bit-wise operations used to shuffle certain fields of the packet header or hash result to cope with hash polarization [20, 24, 31]. Hash polarization would cause load imbalance when hashing results favor certain buckets over others. Pre-processing usually uses bit-wise operations such as AND, XOR, bit shifting and masking. Post-processing uses XOR-folding of 32-bit hash results to get a 16-bit result [38]. Our analysis shows that these bit-wise operations in pre-processing and post-processing functions do not affect the linear property of ECMP path selection. The detailed proof can be found in supplemental materials [48].

Linearity also holds for the $Bucket_B^N()$ function if N and B are both powers of two, which is expected if a fat-tree topology is used. Then the modulo operation is equivalent to a bit-wise shifting, which is proved to be linear. There are two scenarios in which linearity does not apply for ECMP. The first scenario is that the next hop ID of the *Bucket* is randomized. However, we can reorder buckets based on the next hop ID after bucket mapping updates, which is commonly adopted for consistent hashing. The second scenario is that the number of buckets is not a power of two 2^k, where k is an integer. The number of buckets depends on the topology.

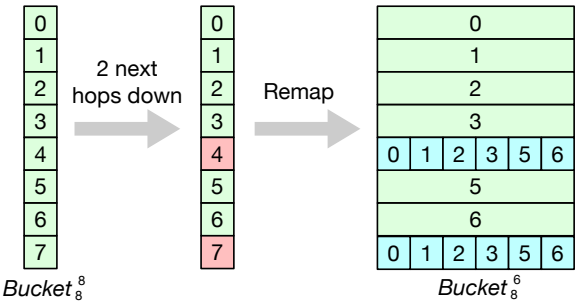


Figure 3: Consistent bucket mapping upon 2 link failures. Buckets of failed hops are remapped to 6 virtual entries.

Table 1: Probability of different failure scenarios with various # of failed links and probability that linearity holds ($\mathbb{P}$) under various link failure ratios σ ($B = 8$)

# failed links	0	1	2	3	$\mathbb{P}$
σ=1%	92.2%	7.45%	0.26%	0.005%	99.68%
σ=0.1%	99.2%	0.79%	0.003%	$< 10^{-9}$	99.90%
σ=0.01%	99.9%	0.08%	$< 10^{-6}$	$< 10^{-10}$	99.99%

If a fat-tree topology is used, the number of buckets is 2^k. If the data center adopts a variation of a fat-tree topology, each switch can enable consistent hashing by adding virtual nodes to make the bucket number as 2^k [33].

Thus we have

$$ECMP(h_j \oplus \Delta) = ECMP(h_i) \oplus ECMP(\Delta) \oplus ECMP(0) \quad (3)$$

where h_i and h_j are arbitrary packet headers, $ECMP(0)$ is the output given an all 0's header, a constant for a given ECMP configuration.

Linearity upon link and device failures: In the case of link and device failures, the number of next hops N varies and may not always be a power of two, which breaks the linearity of the whole ECMP process. We leverage consistent hashing and bucket remapping to guarantee linearity for most links in the ECMP group. We illustrate our ideas with an example with two link failures. In the case of failures, for example both next hop 4 and 7 fail as shown in Figure 3, removing two buckets ($Bucket_6^6$) will break linearity since 6 is not a power of two. If we keep these two buckets and simply adopt consistent hashing to remap these two buckets to two random available next hops, the load is imbalanced given 8 buckets are mapped to 6 hops. We thus remap these two buckets to 6 virtual entries and map these entries to available next hops uniformly as shown in Figure 3. Therefore, we can still

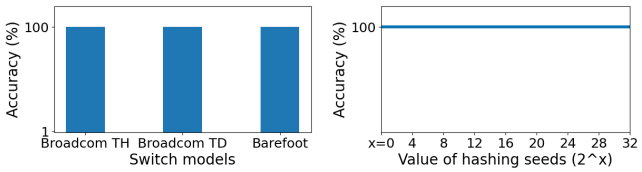
preserve the linearity for available links by fixing the hash bucket B as a power of two. The load balancing performance is equivalent to an ECMP table with size of the least common multiples of B and N. And the $Bucket_B^N()$ function can be modeled as follows:

$$Bucket_B^N() = Bucket_B^B() \; is \; down? \; Bucket_N^N() : Bucket_B^B() \quad (4)$$

where $Bucket_B^B()$ holds linearity as B is a power of two. And the non-linear function $Bucket_N^N()$ is a modulo operation over N followed by a mapping from $\{0,...,N-1\}$ to all available next hop IDs.

As stated in [15], links and devices inside data center networks usually have four 9's of high availability. Before BGP updates bucket mapping upon the first link failure, linearity still holds since the output of Bucket() remains the same. The linearity-based path control might fail upon more failures, but the probability is low. For example, the probability that more than 1 failure happens in the same ECMP group is only 41% [15]. We analyzed the probability that linearity holds under various link failure ratios as shown in Table 1. We consider the probability of occurrence with various numbers of failed links under different link failure ratios. Given F next hops removed due to failure links, $Bucket_B^N()$ holds linearity with a probability of $1 - F/B$. Based on our observation of 1%-0.01% failure rate σ in our production data centers of more than 1500 links, the probability that linearity holds is more than 99% even when link failures are considered. Here we assume link failures are independent. If we assume the probability of link failures under the same ECMP group is positively correlated, the probability of recovery decreases. However, according to previous studies [15] and our observations in our own data centers, the probability of concurrent failures in one ECMP group is low.

Validation: We validate linearity in commercial switches by checking whether Equation 3 holds. We configure the switch with 8 next hops. First, we generate 1 million packet headers with random IP addresses, port numbers, protocol field and reserved fields. For each header, we record the output of ECMP before and after adding the offset Δ, which are denoted as $ECMP(h_i)$ and $ECMP(h_i \oplus \Delta)$, where h_i are random packet headers and Δ ranges from 0 to $2^k - 1$ for a k bit field. We do the same for another random packet header h_j. Finally, as shown in Figure 4, by comparing whether $ECMP(h_i) \oplus ECMP(h_i \oplus \Delta) = ECMP(h_j) \oplus ECMP(h_j \oplus \Delta), \forall i, j$ holds, we verify that switches from Broadcom and Barefoot all satisfy the linearity. We also compare with different hashing seed configurations. The results show that linearity holds for different seed settings from 0 to $2^{32} - 1$. Linearity holds for validated vendors under XOR hashing, CRC hashing, and variants of XOR/CRC hashing (e.g. optimized hashing by concatenating COR hashing results and CRC hashing results). Besides the Broadcom and Barefoot switches that we validated in the paper, switches from 100+ platforms with SAI also support linear hashing [29, 34].



(a) Different vendors (b) Impact of seeds

Figure 4: Validation with commercial switches. Accuracy of ECMP linearity-based prediction is 100% in different vendors and under selection of different seeds.

3.3 The Path Linearity

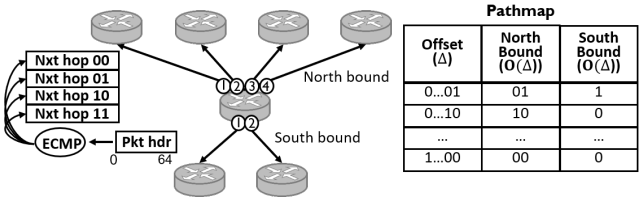
Our previous analysis has demonstrated the linearity of ECMP path selection at a single switch. In this section, we analyze the linearity behavior of ECMP in the multi-hop environment and discuss how the linearity can be used for path control.

Multihop linearity: In the multihop environment, a path is defined as a list of end hosts and switches, for example, $(A \rightarrow L1 \rightarrow S2 \rightarrow C3 \rightarrow S4 \rightarrow L4 \rightarrow B)$ is a path from the end host A to the end host B in Figure 1. In general, a path is modeled as $(sender, s_1, ..., s_X, receiver), s_i \in \{0, ..., N_i - 1\}$, where X is the number of hops, N_i is the number of equal-cost next hops at hop i, and s_i is the next hop ID chosen by hop $i-1$. There can be one or more links between the end host and leaf switches.[1]

We compile the path as the concatenation of the binary representation of next hop selections $Path(h) = s_1 \cdot s_2 \cdot ... \cdot s_X$, where the sender and receiver are omitted. In a typical three-tier data center network, X is 5 for inter-pod traffic and 3 for intra-pod traffic, where a pod is an aggregation of leaf and spine switches as shown in Figure 1. Since ECMP decides the next hop at each switch, we have $s_i = ECMP_{s_{i-1}}(h), i = 1, ..., X$. In order to calculate a path, we need the ECMP functions of all switches along the path. However, since we are only interested in the linearity, we aim to calculate $Path(h \oplus \Delta)$. As our analysis shows, the linearity $ECMP(h)$ holds at a single hop. For two hops $s_1 \cdot s_2$, concatenation of $s_1 = ECMP_{sender}(h)$ and $s_2 = ECMP_{s_1}(h)$ still satisfies $ECMP_{sender}(h \oplus \Delta) \cdot ECMP_{s_1}(h \oplus \Delta) = (ECMP_{sender}(h) \cdot ECMP_{s_1}(h)) \oplus (ECMP_{sender}(\Delta) \cdot ECMP_{s_1}(\Delta)) \oplus (ECMP_{sender}(0) \cdot ECMP_{s_1}(0))$ under the assumption that the selection of s_1 will not affect $ECMP_{s_1}(\Delta)$ and $ECMP_{s_1}(0)$. Note this assumption holds as long as switches in the same ECMP group are configured with the same hashing configuration. By the same methodology, we can prove the linearity for the concatenation of the entire path. Then we have

$$Path(h \oplus \Delta) = Path(h) \oplus Path(\Delta) \oplus Path(0) \quad (5)$$

[1] If there are several leaf switches to choose, the end host usually adopts Link Aggregation Control Protocol (LACP) to choose the next hop based on the hashing result of packet header fields like ECMP [4, 9]. Our analysis on ECMP linearity applies to LACP.



Pathmap

Offset (Δ)	North Bound (O(Δ))	South Bound (O(Δ))
0...01	01	1
0...10	10	0
...	...	...
1...00	00	0

Figure 5: Illustration of pathmap. There are 64 offsets for a header space with 64 bits. Each row represents the path change $O(\Delta)$ for northbound and southbound with offset Δ.

Pathmap: Equation 5 suggests that with the linearity of ECMP path, given a relative change to an arbitrary packet header $(h \oplus \Delta)$, we can predict the corresponding ECMP path change as $Path(h \oplus \Delta) = Path(h) \oplus O(\Delta)$, where $O(\Delta) = Path(\Delta) \oplus Path(0)$. This mapping relationship from Δ to $O(\Delta)$ can be modeled with a pathmap structure. As shown in Figure 5, we use a $K \times 2$ table to represent the pathmap of a single switch, where K is the number of bits used for path control, and each column corresponds with the northbound and southbound path offset $O(\Delta)$ for corresponding header offset Δ. If there are N hops, we need a $K \times N$ table. Here we use K entries since we can represent the entire header space by XORing Δ, where Δ are headers 0...010...0 with only one significant bit. With this model, we compress the ECMP model exponentially from the header space 2^K. In most well-defined packet headers, the number of bits that can be used for path control is limited, which restricts the size of the pathmap.

Insights: Given the proven linearity, we can control the relative path change by changing certain header fields. This is feasible since hashing linearity is a built-in property for linear hashing algorithms. To extend hashing linearity to path linearity and improve accuracy upon link failure, we also need to adopt several ECMP hashing configurations, i.e. consistent hashing, bucket remapping and the same hashing configuration for the same ECMP group[2].

4 Relative Path Control

4.1 Design

Our analysis of the ECMP hash linearity suggests a deterministic mapping between the packet header changes and path changes. It provides a powerful tool to predict the relative path offset $O(\Delta)$ based on a packet header offset Δ.

We propose a relative path control algorithm, `RePaC`, based on the insight from hashing linearity. `RePaC` has two parts: offline pathmap collection and online path control. The offline pathmap collection module acquires the mapping from relative header change to the relative path change based on the static configuration of the network. The online module

[2]Supplemental materials [48] discuss a solution with relaxed requirements that allow different hashing seeds.

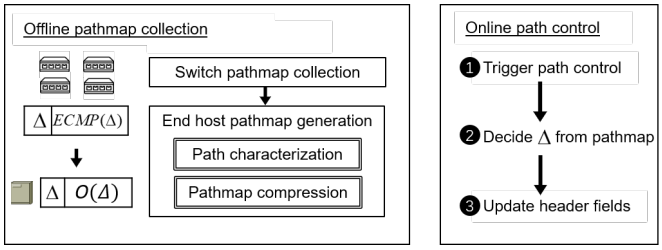


Figure 6: Relative Path Control Process

decides the header change accordingly and alters packets to navigate certain flows to a different path with deterministic offset. The overall procedure is shown in Figure 6.

Offline pathmap collection: In order to collect the pathmap for a switch as shown in Figure 5, we need to send the packets with reference headers with different offsets Δ to northbound and southbound egress ports on the switch. As introduced in Section 3.3, Δ is 0...010...0 with a single significant bit, and the position of the significant bit ranges from 1 to K, where K is the number of controllable bits in the packet header. The pathmap of a switch depends on the static ECMP hash configuration. So to collect the pathmap for a whole data center network, we need to consider different hashing algorithms (e.g. XOR, CRC with different polynomials) and different hashing seeds. In production data centers, the switches in the same tiers of the network are often designed to use the same ECMP configurations for the simplicity of network management and operations. Then we only need to collect pathmaps for three types of ECMP hash configurations for a typical data center network in a 3-tier Clos topology. Note if the hashing algorithm is fixed, the mapping between the hashing seed and pathmap is deterministic, which reduces the overhead of collecting pathmaps from switches with different hashing seeds. The offline pathmap collection can be easily done on today's network management systems.

Online path control: The online path control module has two parts, the triggering function and the decision function. The triggering function is triggered when path control is needed, for example, upon failures. The decision function decides the path header offset. Users can design the decision function based on application requirements. We define a utility function $util(\Delta)$ for path control. The target of path control is $\hat{\Delta} = argmax(util(\Delta))$. For example, if the target is to change the path, we define $util(\Delta) = Path\_changed(\Delta)$, where $Path\_changed(\Delta)$ is a function to check whether applying header offset Δ could change the path. We showcase how to design the utility function in Section 4.2.

Based on the designed utility function, the pathmap can be reconstructed in different formats to facilitate searching for an offset for a specified relative path change. For example, a pathmap can be compressed into a hashmap with keys as desired path changes and values as all possible header offsets. For solutions that are not sensitive to the value of relative path change, for example, a failover solution that aims to use

a specific path change, the pathmap can be compressed to a list of desired header offsets that can lead to the desired path change. The structure of the pathmap is decided by the granularity of path control required by applications.

Another key question is which header fields to use for path control. Options depend on what user-defined fields are configured for ECMP hashing. By default, switches might only configure IP addresses and port numbers for ECMP hashing. Based on [28], SAI-supported switches can configure all header fields for hashing. Among all the fields, the high significant bits of TTL fields (8 bits) are typically usable since the number of hops in data centers is small. If switches do not support user-defined fields, the source port number (16 bits) can be leveraged though it might require changes of the port allocation. Besides, the source IP address can be controlled with certain flexibility in incast scenarios if there is a configurable DHCP server inside the pod to assign IP addresses dynamically.

4.2 The Applications of Relative Path Control

In this section, we present two case studies to demonstrate the applications of relative path control.

4.2.1 RePaC for fast failover

Leveraging relative path control, we design a lightweight and fast failover mechanism to detect path failures at the transport layer and update the packet headers to migrate the traffic from the failed path. Our approach relates to the previous end-host-based failure recovery [19,23] but gives strong guarantees on path control with the ECMP hashing linearity and does not require any changes to the network protocols and switch hardware. We demonstrate a failover mechanism with TCP traffic as an example. The RePaC-based failover mechanism generally applies to any network transport layer protocol with a retransmission mechanism.

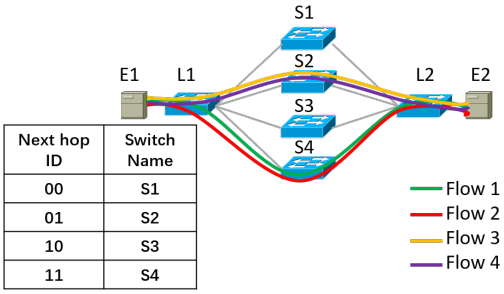
The RePaC-based fast failover involves retransmission-triggered failure detection and failure recovery by altering packet headers. It works as shown in Algorithm 1. Our algorithm detects the second retransmission of each flow to trigger failure recovery, then every packet in that flow will be marked for path control. We choose the higher four bits of TTL fields and TCP five tuples as input for $ECMP()$. In this case, we can vary the higher four bits of TTL fields, so Δ is in $\{0001, 0010, 0100, 1000\}$.

After failure recovery is triggered, we mark the reserved bits with a different value to reroute this packet to a different path. In theory, if the path offset $O(\Delta)$ is non-zero at each hop, we can ensure the new path $Path(h \oplus \Delta)$ has no overlapping links or devices with $Path(h)$, where h is the packet header of the flow affected by the failure. With RePaC, we can generate a path mapping table from all marking values to the network path change, as illustrated in Figure 5. And we design the utility function as $util(\Delta) = Path\_changed(\Delta)$, where $Path\_changed(\Delta)$ is a simple function to check whether $O(\Delta)$ is non-zero for each hop, which indicates that the next hop selected at each hop would be different from the previous path. Then we use a subset of eligible Δ as marking values for fast failover purposes. If we cannot find an $O(\Delta)$ with all hops changed, we select the one with the most changes. RePaC will stop path marking after a configurable timer. We set the timer as the failure detection time in BGP. Path marking will be triggered again if BGP fails to recover the failure.

Algorithm 1 Failure recovery

Input: Packet p and corresponding flow f
Output: Packet p with f.mark within path marking period. Otherwise, original packet p

```
1: if p is a repeated retransmission then
2:     Update f.mark = SelectNextMark(p)
3:     Begin path marking period
4: end if
```



Next hop ID	Switch Name
00	S1
01	S2
10	S3
11	S4

Figure 7: Example network. Flow polarization causes under-utilization of paths.

4.2.2 RePaC for traffic engineering

Given the multi-path nature of data center networks, traffic engineering, the ability to schedule traffic on many non-overlapped paths to better leverage the network bandwidth, has been a key part of data center network design [2,5,30,37]. Compared to existing proposals, such as Hedera [2], MicroTE [5], and XPath [21], RePaC provides a lightweight tool for us to plan and distribute flows in data center networks for traffic engineering without any changes to the switch hardware and routing protocols.

Path planning using RePaC: Given the topology and ECMP configurations of the network, we derive the pathmap M, a map between each path offset $O(\Delta)$ and the corresponding header offset Δ, by sending probing packets with different Δ. Here we show an example for the network in Figure 7. We consider the lower 6 bits of *src_port* is controllable and probe switch L1 for $O(\Delta)$ to get Table 2. Table 2 shows how changing a bit in *src_port* changes the selected hop ID. XORing all possible hop ID changes results in four different path offsets.

Table 2: Probed results. Six controllable bits generate six headers Δ, each with one significant bit. Then RePaC uses six headers to get $O(\Delta)$.

Δ	000001	000010	000100	001000	010000	100000
$O(\Delta)$	00	00	01	00	01	10

Table 3: Calculated $M = (O(\Delta), \Delta)$ for path planning. For each $\Delta = 0, ..., 63$, RePaC calculates the Path ID.

$O(\Delta)$	0	1	2	3
Δ	0-3, 8-11, 20-23, 28-31	4-7, 12-19, 24-27	32-35,40-43, 52-55, 60-63	36-39, 44-51, 56-59

We then derive all the possible header changes for each path offset $O(\Delta)$. For example, $\Delta = 000100 \oplus 100000$ is mapped to path offset $01 \oplus 10 = 11$. We index these path offsets as Path IDs to differentiate paths for path planning. Finally, We get the pathmap M as shown in Table 3, which is a map from Path ID $O(\Delta)$ to a set of Δ sharing the same path offset $O(\Delta)$.

From Table 3, we can observe that the selected Path IDs are limited to two choices for *src_port* values from 0 to 31. This means if E1 sends flows with *src_port* ranges from 0 to 31, ECMP will select only half of the next hops. We conjecture that only a subset of bits in the hash results are selected by the post-processing function for final ECMP resolution. This suggests that the default ECMP path selection could result in only half of the paths being utilized in the example scenario. Based on the pathmap M in Table 3, we can assign *src_port* values (e.g. i, $i \oplus 4$, $i \oplus 32$, $i \oplus 36$) for those 4 flows to ensure that all four possible paths are utilized.

Based on the analysis in Section 3.3, we can infer the path offset between two flows by looking up the Δ of their packet headers in the pathmap. Therefore, we can use a valid packet header h_{ref} as a reference, then for each $flow_i$, look up the desired path offset in the pathmap for unused Δ and assign $h_{ref} \oplus \Delta$ as packet header h_i. We plan the paths for all flows based on estimated loads as shown in Algorithm 2. In the example network of Figure 7, flows might conflict with each other according to default ECMP. With our algorithm, flows are evenly distributed on diverse paths. Note we can extend the algorithm to consider both the request load and the response load. The pathmap at the receiver can predict the path for response load.

5 Evaluation

5.1 Implementation and Test-Bed Setup

We implement RePaC as a library on the servers to perform on-demand path control. There are three components in the implementation, a pathmap collector that generates a pathmap database given a network, a module that monitors all outgoing

Algorithm 2 Load based path planning

Input: Pathmap M as illustrated in Table 3,
Output: assign a packet header h_i for $flow_i$ to minimize the deviation of bandwidth utilization L_p for each path

1: $h_{ref} \leftarrow$ a random valid packet header; $L_p \leftarrow 0$ for all paths
2: **for** $flow_i$ in all flows **do**
3: Find $\Delta \in M$ for Path ID p with the minimum load L_p
4: $h_i = h_{ref} \oplus \Delta$
5: L_p += estimated bandwidth utilization for $flow_i$
6: **end for**

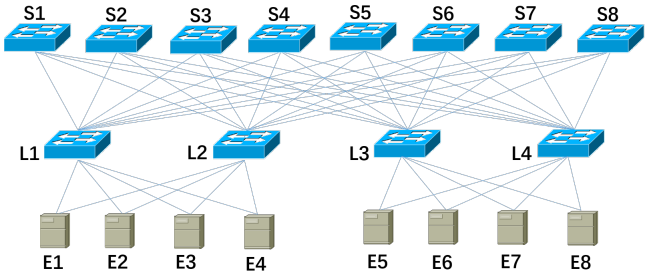


Figure 8: Test topology

traffic on servers to perform TCP failover, and a path planner application that generates the best paths for applications. We implement flow monitoring with the VNIC, which maintains a soft stateful flow table. For each TCP flow, RePaC checks for packet retransmissions and marks packets to perform traffic failover as discussed in Algorithm 1. The total software modules are implemented in 1066 lines of C code. The RePaC library is a lightweight software module that can be easily installed on servers.

We evaluate the performance of RePaC in a fat-tree topology as shown in Figure 8. In this testbed, all the links are 25Gbps. 8 spine switches (Broadcom Trident 3) interconnect 4 leaf switches (Broadcom Tomahawk 3). We run SONiC [35] (an open-source network operating system) on all switches and adopt different ECMP configurations at each tier to avoid traffic polarization. Four physical servers are connected to leaf switches L1 & L2, and another four are connected to L3 & L4. With each server connected to two switches, the server can still connect to another leaf switch if one fails. Each server is equipped with Linux 3.10 or Linux 4.19 and TCP NewReno. These settings are consistent with our production data center networks.

5.2 Effectiveness and Overhead

Path control upon link and device failures: We first evaluate the accuracy of path control under various failure scenarios. Link and device failures are injected based on failure statistics collected in production data centers during one week. We gauge the accuracy of path control by calculating the percentage that the selected path is the expected path. We test with various numbers of spine switches to validate the perfor-

Table 4: Accuracy of path control under failures

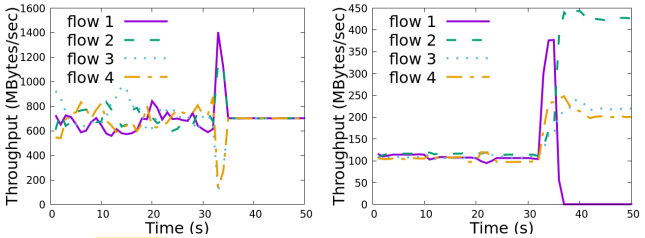
# of spine switches	4	8	16	32
RePaC w\o failures	100.0%	100.0%	100.0%	100.0%
RePaC w\ failures	97.0%	98.5%	99.24%	99.62%
Random w\o failures	25%	12.5%	6.3%	3.1%

mance under different network scales. As shown in Table 4, the accuracy increases with the number of spine switches since the proportion of failed links of total links decreases. As discusses in Section 3.1, RePaC can provide accurate path control for the first link failure if RePaC is triggered before BGP changes the bucket. But RePaC might not work at the first attempt if there are two and more link failures in the same ECMP group. Table 4 shows RePaC provides >97.0% accuracy while the random path selection can only provide <25% .

Overhead: We test the CPU overhead and the memory cost of RePaC with 1 thousand to 1 million concurrent connections. The average CPU cost on a 64-core processor is about 0.01% for 1k connections and 0.05% for 1m connections. Note that the performance of RePaC is independent of the kernel version because RePaC is lightweight and implemented in a portable VNIC module. At each server, we allocate about 64 bytes of extra memory for each connection to keep track of its state. We also evaluate whether RePaC reduces packet processing speed. RePaC looks up the flow table on every packet, checks packet sequence number, and updates the flow state accordingly. These bring a few extra memory accesses to the data path. We observe no degradation in packet processing speed with different workloads.

5.3 RePaC for Failover Evaluation

Experiment methodology: We study the failover performance of RePaC by measuring throughput and downtime under common failure scenarios. We run a high-priority application on the VNIC interface to collect throughput measurements at a 20ms interval, which also helps us determine the downtime upon failure. We consider two failure scenarios, silent drop failures and link failures. We simulate silent drop failures by configuring an ACL rule to discard the data traffic. We simulate link failures by issuing ifconfig down command to deactivate one of the interfaces from receiving and sending data from switch CLI. There are many existing studies on failover [19, 23, 43, 44, 50]. Among them, MPTCP is a fair baseline as an end-host-based mechanism. MPTCP employs a group of subflows with random paths for failover without adding any centralized controller or advanced hardware. We use MPTCP v0.90 with the redundant scheduler for comparison with RePaC. We test MPTCP with the redundant scheduler over the default scheduler since it provides better fault tolerance by sending redundant copies through all subflows.



(a) With RePaC (b) with MPTCP

Figure 9: Throughput during failure

Failure recovery performance: We compare RePaC with both end-to-end and in-network failover design. Compared with end-to-end failover MPTCP, which recovers with a random backup path, RePaC provides better recoverability. Compared with in-network failover approaches, RePaC covers more failure types and reduces downtime by orders of magnitude.

We first compare RePaC with MPTCP in Figure 9. Figure 9a shows an example trace the silent drop on leaf switch L1 get recovered after silent drop failure with RePaC. Flow 3 and flow 4 are forwarded to L1 by ECMP and hence dropped by ACL. Both flows experience around 1 second of disruption and then are re-routed to switch L2. Flow 1 and flow 2 forwarded by switch L2 only experience oscillation. During the oscillation, the throughput of flow 1 and flow 2 first doubles when the other two flows stop transmission, then returns to similar throughput as before. All flows eventually share the bandwidth of the bottleneck link fairly with congestion control. Note that the bottleneck is the L3-E5 link, so the throughput before and after the failover are similar. In comparison, Figure 9b shows that MPTCP failover performance is unsatisfactory. We configure each flow with 4 subflows to increase the path diversity. Figure 9b shows that MPTCP can recover all affected flows except flow 1 because all subflows of flow 1 are dropped. The throughput of flows 2-4 increases after flow 1 is deactivated. Overall throughput performance with MPTCP is worse than RePaC since MPTCP with the redundant scheduler copies packets on all available subflows.

We further compare the failure recovery ratio between RePaC and MPTCP with different numbers of subflows for MPTCP. We define the recovery ratio as the probability that a flow affected by failure is recovered. As shown in Figure 10, MPTCP offers a higher recovery ratio with more subflows. Although MPTCP provides a reasonable recovery ratio with the redundant scheduler, RePaC performs better by leveraging path diversity with the pathmap. RePaC outperforms MPTCP with 2(4) subflows by 36%(21%). RePaC outperforms MPTCP because MPTCP cannot guarantee there exists at least a subflow unaffected by the failure.

We also compare RePaC with in-network failover mechanisms under various failure scenarios. As shown in Figure 11a, we start a single TCP flow from E1 to E5 and measure the downtime after simulating a device failure at L1. We simulate the device failure by deactivating all the interfaces. In prac-

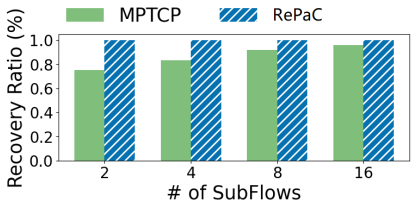
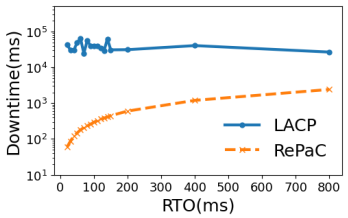
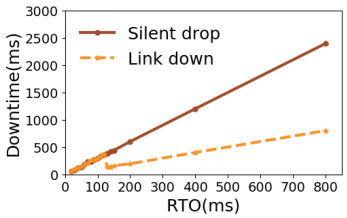


Figure 10: Recoverability comparison between RePaC and MPTCP.



(a) Comparison between RePaC and in-network failover under device failure



(b) RePaC with in-network failover under slient drop and link down

Figure 11: Downtime comparison under various minimum RTOs

tice, device failures might be triggered by hardware failure, unexpected reboot, or power loss. Although this type of failure can be detected by Link Aggregation Control Protocol (LACP) using heartbeat signals, it takes up to 90 seconds. In comparison, RePaC significantly reduces the downtime by orders of magnitude. Note the recovery Algorithm 1 initiates path control after the second TCP retransmission. Our solution performs better with a smaller minimum RTO. RePaC reacts to failures in around 60ms with a 20ms minimum RTO. Theoretically, RePaC can provide sub-ms recovery if the end hosts adopt high-resolution timers.

In Figure 11b, we evaluate how our solution can cooperate with the existing in-network failover approaches used in data centers, such as link scan, Bidirectional Forwarding Detection (BFD), LACP, and BGP. We compare the downtime caused by two types of failures, silent drop failures and link down events. For silent drop failures, our design can recover affected flows within three times of RTO, as shown in the solid line in Figure 11b. After a link down failure is injected, RePaC is triggered faster than in-network approaches when the second retransmission is triggered before the shortest in-network failover timers (130 ms for the link scan function). Note RePaC kicks in after around the triple of the minimum RTO. When the minimum RTO increases, RePaC will not be triggered since the link scan function would recover failures faster. In summary, Figure 11 proves RePaC can cooperate well with existing failover mechanisms and help to reduce the flow downtime.

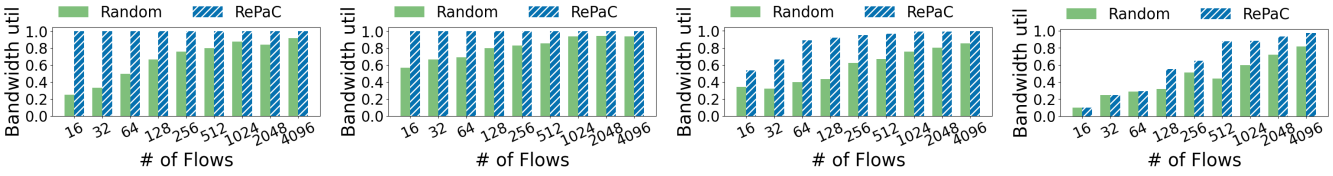
5.4 RePaC for Traffic Engineering Evaluation

Experiment methodology: We gauge the performance of load balancing and flow completion time of RePaC compared with random path selection, precise path control via XPath [21], and in-network load balancing via CONGA [3]. We evaluate RePaC for load balancing under two scenarios: one-to-one flows and many-to-one incast flows. For one-to-one flows, we vary the *src_port* to vary the path selection of these parallel flows. For many-to-one incast flows, we vary *src_ip* and *src_port* for four servers under the same rack. We assume the source port number is flexible for traffic engineering applications. For many-to-one flows, we assume the source IP address is also assignable. We vary the least significant 8 bits of *src_ip* inside a cluster to improve path diversity. We test with three representative traffic patterns, same-size flows, webserver flows, and Hadoop flows. Webserver flows follow a uniform size distribution, and Hadoop flows follow a skewed size distribution [40].

Load balance performance: In Figure 12, we compare the bandwidth utilization of RePaC path planning with random path planning. The bandwidth utilization is measured using the average utilization ratio of the links in the network. For one-to-one flows, as seen in Figure 12a, RePaC increases the bandwidth utilization by up to 3 times when the total number of flows is 16. For many-to-one connection, Figure 12b shows the bandwidth utilization of links when flows of the same size from different sources are sent to a single destination. Our solution can evenly distribute the flows across different paths, achieving well-balanced bandwidth utilization. For the webserver case, as shown in Figure 12c, the bandwidth utilization increases from 0.54 to 0.99 using RePaC, compared to 0.34 to 0.85 with random selection as the number of flows increases from 16 to 4096. Figure 12d shows the bandwidth utilization for Hadoop with skewed flow size distribution. Both algorithms show low bandwidth utilization when the number of flows is small. When more flows join, RePaC can 20% to 50% better bandwidth utilization.

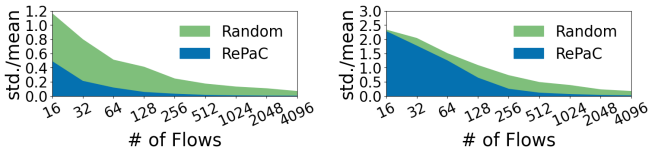
In Figure 13, we compare our solution with random path selection by measuring the load on each path when the number of concurrent flows increases from 16 up to 4096. Here we calculate the load at each path and plot the standard deviation over mean in Figure 13. As the number of flows increases, RePaC provides much better performance. For webserver traffic, the std./mean reduce by 58% to 97% as shown in Figure 13a. Our design tends to select a path with the least load for each flow. In Figure 13b, when the number of flows is greater than 64, the std./mean is reduced by more than 41%.

For time-sensitive applications, such as distributed file systems (DFS), flow completion time is crucial. We simulate DFS by sending 100GB of data from one host to another. We compare the completion time of all concurrent flows between RePaC and random path selection. RePaC can plan paths based on the traffic load, so elephant flows are distributed to separate paths. We notice that the average completion time of our solution is considerably less than the random path selection as shown in Figure 14a. When there are 16 flows, the comple-



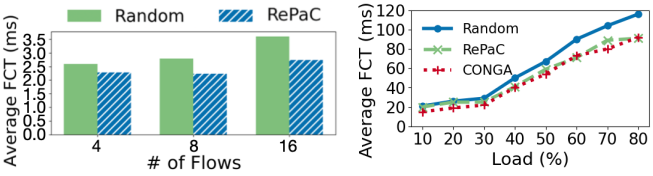
(a) One-to-one connection (b) Many to one - flows of the same size (c) Many to one - web server (d) Many to one - Hadoop

Figure 12: Link bandwidth utilization of path planning.



(a) Web server (b) Hadoop

Figure 13: Load balance of path planning. The shading area shows the std./mean. of load on each switch. Larger shading area means load is more unbalanced.

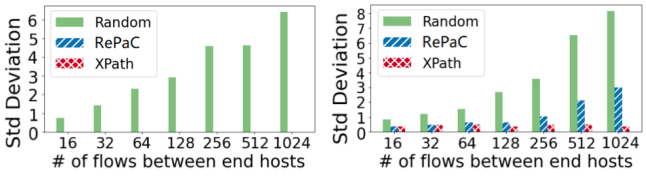


(a) File distribution (b) Short requests

Figure 14: Flow completion time.

tion time reduces by 25% from 3.6 seconds to 2.7 seconds. Besides large file distribution, we also simulate short requests and gauge the flow completion time compared with random path selection and CONGA [3]. We use simulation since CONGA requires advanced hardware. Each host will inject flows with empirical traffic distributions of short requests into different queues based on the given path selection algorithm. CONGA will measure the queuing time of each path and choose the path with the least queuing time to send flowlets, thus performs best. RePaC tends to select the path with the least number of concurrent flows. As shown in Figure 14b, RePaC's performance is comparable with CONGA.

Load balance performance under topology updates: The network topology might change if network devices are removed for maintenance or upgrade. In Figure 15, we compare the load balance performance between random path selection, RePaC, and XPath [21] when the network topology changes. Since XPath is based on source routing, which limits its deployment on our testbed, we use simulation with the same testbed topology. We simulate multiple concurrent flows between end-hosts and compare load balance by the standard deviation of the load on each available path. XPath relies on a centralized controller to perform topology updates, which, we assume, can provide accurate path estimation. As shown in Figure 15a, without topology updates, both RePaC and XPath distribute the flows evenly. After removing a spine



(a) w/o topology updates (b) w/ topology updates

Figure 15: Load balance performance when topology changes

switch, as shown in Figure 15b, XPath can still balance the flows using accurate path information. RePaC can distribute most flows evenly on the unchanged paths using outdated pathmaps. In conclusion, XPath has the best performance leveraging pre-installed routes and real-time path information updates. RePaC degrades because of the outdated pathmap. However, both outperform the random path selection significantly. Compared with XPath, RePaC is a lightweight solution that significantly outperforms the random scheme without redesigning the routing protocols .

6 Discussion

Our work is the first study that performs a detailed analysis on the ECMP hashing linearity and its applications to path control in data centers. More work could be done to further explore the design of multi-path hashing mechanisms and their implications to data center networks.

ECMP configuration optimization: Our analysis has demonstrated that the properties of ECMP hashing algorithms have significant impacts on traffic engineering and failover in data center networks. However, ECMP configurations in today's production data centers are still done by operators using ad-hoc approaches. Future investigations of ECMP might reveal other interesting properties rather than linearity, which could facilitate the data center network optimization and result in a more scientific approach to network-wide configuration optimization.

Flow analysis based on hash simulation: Analysis of the ECMP path selection algorithm also sheds insights on how flows are distributed. Using real-time network topology information, network operators could build a lightweight simulation network with the model of ECMP hashing behaviors. The simulation network can answer many what-if questions about the network performance and flow distributions, such as how many flows will be affected by certain failure events

or a congested link.

Programmable hashing on switch ASICs: Today's data center switch ASICs provide limited interfaces to configure ECMP hash algorithms and parameters. Even on programmable switch ASICs (e.g. Tofino), where users can program the packet processing pipeline with network programming languages, there is still limited programmability to define ECMP hashing behaviors. Given that the hash algorithm could significantly impact the overall networking performance, a programmable interface that allows users to redesign the hash algorithms could lead to more innovations in this space.

RePaC capable applications: RePaC provides a lightweight mechanism for applications to predict and control the paths of packet flows in the network, without the requirements for deploying any new protocols, centralized controllers, or advanced hardware. Many applications can benefit from path diversity. For example, network probing tools, such as PingMesh [17], can guarantee full network coverage while minimizing the probing overhead with the help of pathmap. The same benefits apply to applications with low latency and high throughput requirements, such as distributed file systems or AI training clusters. RePaC provides a powerful tool for the applications to schedule flows to avoid traffic congestion and disruption.

RePaC and future ECMP: RePaC requires hashing linearity, which is a built-in feature for linear hashing algorithms. For future hashing algorithms, there is no conclusion on whether linear hashing is preferred or non-linear hashing. Existing ASICs combine XOR and CRC to improve hashing while keeping linearity [33]. We are not aware of any studies on the possible tradeoff between linearity and hashing performance. It is still unclear whether non-linear hashing definitely outperforms all variants of linear hashing. Lightweight path control for future variants of hashing algorithms remains an interesting research problem. In order to support RePaC, ECMP can only adopt linear hashing algorithms. We expect future standardizations of ECMP to include linearity as another metric to consider besides hashing performance.

7 Related Work

RePaC is related to several prior lines of work:

End-host based path control: There are two spectrums of end-host-based path control. The first spectrum of approaches gets rid of ECMP and redesigns the routing protocols in the network [21, 22, 39, 41]. XPath is the most recent work, which identifies and pre-installs routes on switches [21]. Though XPath reduces the routing table storage with compression algorithms, the communication between end-hosts and the path manager under topology changes and link failures incurs too much overhead. MPLS is the common practice for traffic engineering in core networks [39, 41]. However, MPLS is not suitable for data centers since it can only support a limited number of tunnels. The OpenFlow-based solution relies on on-chip forwarding rules and compresses forwarding rules with a tiny flow table [22]. However, this approach does not apply to data centers due to too many flows. Fundamentally, these approaches require excessive routing/forwarding tables because of the large scale of header space and the variability of paths.

Another spectrum still keeps ECMP as the path selection in switches but extracts the path selection model. Volur [47] attempts to provide path control with a centralized controller that aggregates routing information from in-network switches and disseminates routing information to end-hosts. However, the path selection model incurs too much overhead since it uses a mapping from the entire header space to all possible paths. The header space grows exponentially to the number of bits in header fields used by ECMP. We argue that explicit path control is not necessary so that we can reduce the complexity of the path selection model. Compared with this spectrum of work, we leverage the linearity of ECMP path selection and reduce the model complexity significantly.

Failover and traffic engineering: While our work is closely related to prior studies on failover [14, 19, 23, 25, 43, 44, 50] and traffic engineering [2, 3, 5], it is also fundamentally different. First, RePaC enables path control, which is different from the random path selection scheme in [19, 23, 44]. Second, RePaC is lightweight and easy to deploy, while existing approaches rely on advanced programmable switches [3, 14, 19, 43, 44, 50], switch redesigning (e.g. installing Openflow) [5], or a centralized scheduler to get path information [2]. Third, RePaC enables data traffic based path discovery, which outperforms traceroute traffic based approach [25] since the header fields in original data traffic and traceroute traffic are different.

8 Conclusion

In this paper, we analyze the ECMP hash algorithms used in today's data center switch ASICs. Our analysis shows that the hash algorithms (e.g. XOR and CRC) used in the most popular switch ASICs on the market maintain linearity. We analyze how linearity sheds insights on relative path control. We design RePaC to leverage linear property and show that RePaC can be used in two representative applications, faster failover and traffic engineering. Through extensive evaluation in production data centers, we show that RePaC is easy to deploy and benefits failover and traffic engineering.

Acknowledgments

We would like to thank our shepherd, Amy Ousterhout, and the anonymous reviewers for helping us improve this paper. We would also like to thank the anonymous referees for their valuable comments and helpful suggestions on earlier versions of this paper.

References

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74, 2008.

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

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

[4] Arista. Port channels and lacp. https://www.arista.com/assets/data/pdf/user-manual/um-eos/Chapters/Port%20Channels%20and%20LACP.pdf, 2019.

[5] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, New York, NY, USA, 2011. Association for Computing Machinery.

[6] Broadcom. Broadcom-network-switching-software. https://github.com/Broadcom-Network-Switching-Software/SDKLT/blob/7a5389c6e0dfe7546234d2dfe9311b92b1973e7b/src/bcmptm/include/bcmptm/bcmptm_rm_hash_internal.h, 2019.

[7] Emily Carr. Why merchant silicon is taking over the data center network market. https://www.datacenterknowledge.com/networks/why-merchant-silicon-taking-over-data-center-network-market, 2019.

[8] Guo Chen, Yuanwei Lu, Yuan Meng, Bojie Li, Kun Tan, Dan Pei, Peng Cheng, Layong (Larry) Luo, Yongqiang Xiong, Xiaoliang Wang, and Youjian Zhao. Fast and cautious: Leveraging multi-path diversity for transport loss recovery in data centers. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 29–42, Denver, CO, June 2016. USENIX Association.

[9] Cisco. Understanding etherchannel load balancing and redundancy on catalyst switches. https://www.cisco.com/c/en/us/support/docs/lan-switching/etherchannel/12023-4.html, 2007.

[10] Charles Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(2):406–424, 1953.

[11] M. Davies. Traffic distribution techniques utilizing initial and scrambled hash values, 2010. US Patent US7821925.

[12] Dell. Dell configuration guide for the s4048-on system 9.9(0.0) ecmp-rtag7. https://www.dell.com/support/manuals/us/en/19/force10-s4048-on/s4048_on_9.9.0.0_config_pub-v1/rtag7?guid=guid-9bda04b4-e966-43c7-a8cf-f01e7ce600f4&lang=en-us, 2019.

[13] Nathan Farrington and Alexey Andreyev. Facebook's data center network architecture. In *2013 Optical Interconnects Conference*, pages 49–50. Citeseer, 2013.

[14] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 225–238, 2017.

[15] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 conference*, pages 350–361, 2011.

[16] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 63–74, 2009.

[17] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 139–152, 2015.

[18] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review*, 45(4):465–478, 2015.

[19] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. Blink: Fast connectivity recovery entirely in the data plane. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*,

pages 161–176, Boston, MA, February 2019. USENIX Association.

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

[21] Shuihai Hu, Kai Chen, Haitao Wu, Wei Bai, Chang Lan, Hao Wang, Hongze Zhao, and Chuanxiong Guo. Explicit path control in commodity data centers: Design and applications. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 15–28, Oakland, CA, May 2015. USENIX Association.

[22] Sangeetha Abdu Jyothi, Mo Dong, and P. Brighten Godfrey. Towards a flexible data center fabric with source routing. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, New York, NY, USA, 2015. Association for Computing Machinery.

[23] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, page 149–160, New York, NY, USA, 2014. Association for Computing Machinery.

[24] M. Kalkunte. High speed trunking in a network device, 2005. US Patent US20060114876A1.

[25] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. Clove: Congestion-aware load balancing at the virtual edge. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, page 323–335, New York, NY, USA, 2017. Association for Computing Machinery.

[26] Parantap Lahiri, George Chen, Petr Lapukhov, Edet Nkposong, Dave Maltx, Robert Toomey, and Lihua Yuan. Routing design for large scale data centers. *NANOG 55*, 2012.

[27] Petr Lapukhov, Ariff Premji, and Jon Mitchell. Use of bgp for routing in large-scale data centers. *Internet Requests for Comments RFC Editor RFC*, 7938, 2016.

[28] Guohan Lu. Sai hash enhancement with user defined field. https://github.com/opencomputeproject/SAI/blob/master/doc/SAI-Proposal-13-Hash-UDF.md, 2016.

[29] Guohan Lu and Xin Liu. Sai update and look forward. https://www.opencompute.org/files/OCP2018-SAI-Engineering-Talk-MSFT-final.pdf, 2018.

[30] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. Multi-path transport for {RDMA} in datacenters. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 357–371, 2018.

[31] Brad Matthews and Puneet Agarwal. Flow based path selection randomization, 2013. US Patent US8503456.

[32] Liron Mula, Gil Levy, and Aviv Kfir. Using consistent hashing for ecmp routing, 2017. US Patent US9853900B1.

[33] Cumulus networks. Equal cost multipath load sharing - hardware ecmp. https://docs.cumulusnetworks.com/cumulus-linux/Layer-3/Equal-Cost-Multipath-Load-Sharing-Hardware-ECMP, 2019.

[34] opencomputerproject. Sai. https://github.com/opencomputeproject/SAI/blob/484b8b150e53b9a818af9a850d4a78581ca7e9bc/inc/saiswitch.h#L188, 2019.

[35] opencomputerproject. Sonic. https://azure.github.io/SONiC/, 2019.

[36] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath tcp. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 266–277, New York, NY, USA, 2011. ACM.

[37] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? designing and implementing a deployable multipath TCP. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 399–412, San Jose, CA, April 2012. USENIX Association.

[38] Jarno Rajahalme. Performing a finishing operation to improve the quality of a resulting hash, 2014. US Patent US10193806B2.

[39] Eric Rosen, Arun Viswanathan, Ross Callon, et al. Multiprotocol label switching architecture. 2001.

[40] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 123–137, New York, NY, USA, 2015. ACM.

[41] Martin Suchara, Dahai Xu, Robert Doverspike, David Johnson, and Jennifer Rexford. Network architecture

for joint failure recovery and traffic engineering. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '11, page 97–108, New York, NY, USA, 2011. Association for Computing Machinery.

[42] Kovsky T., J. Tsai, and Joe Chang. High performance crc calculation method and system with a matrix transformation strategy, 2003. US Patent US7219293B2.

[43] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Distributed network monitoring and debugging with switchpointer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 453–456, Renton, WA, April 2018. USENIX Association.

[44] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 407–420, Boston, MA, March 2017. USENIX Association.

[45] WiKipedia. Computation of cyclic redundency checks. https://en.wikipedia.org/wiki/Computation_of_cyclic_redundancy_checks, 2019.

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

[47] Qiao Zhang, Danyang Zhuo, Vincent Liu, Petr Lapukhov, Simon Peter, Arvind Krishnamurthy, and Thomas E. Anderson. Volur: Concurrent edge/core route control in data center networks. *CoRR*, abs/1804.06945, 2018.

[48] Zhehui Zhang, Haiyang Zheng, Jiayao Hu, Xiangning Yu, Chenchen Qi, Xuemei Shi, and Guohui Wang. Supplementary material to hashing linearity enables relative path control in data centers. https://zhehuizhang.github.io/files/atc21_sup.pdf, 2021.

[49] Zhiruo Cao, Zheng Wang, and E. Zegura. Performance of hashing-based schemes for internet load balancing. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 1, pages 332–341 vol.1, March 2000.

[50] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. Understanding and mitigating packet corruption in data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 362–375, 2017.